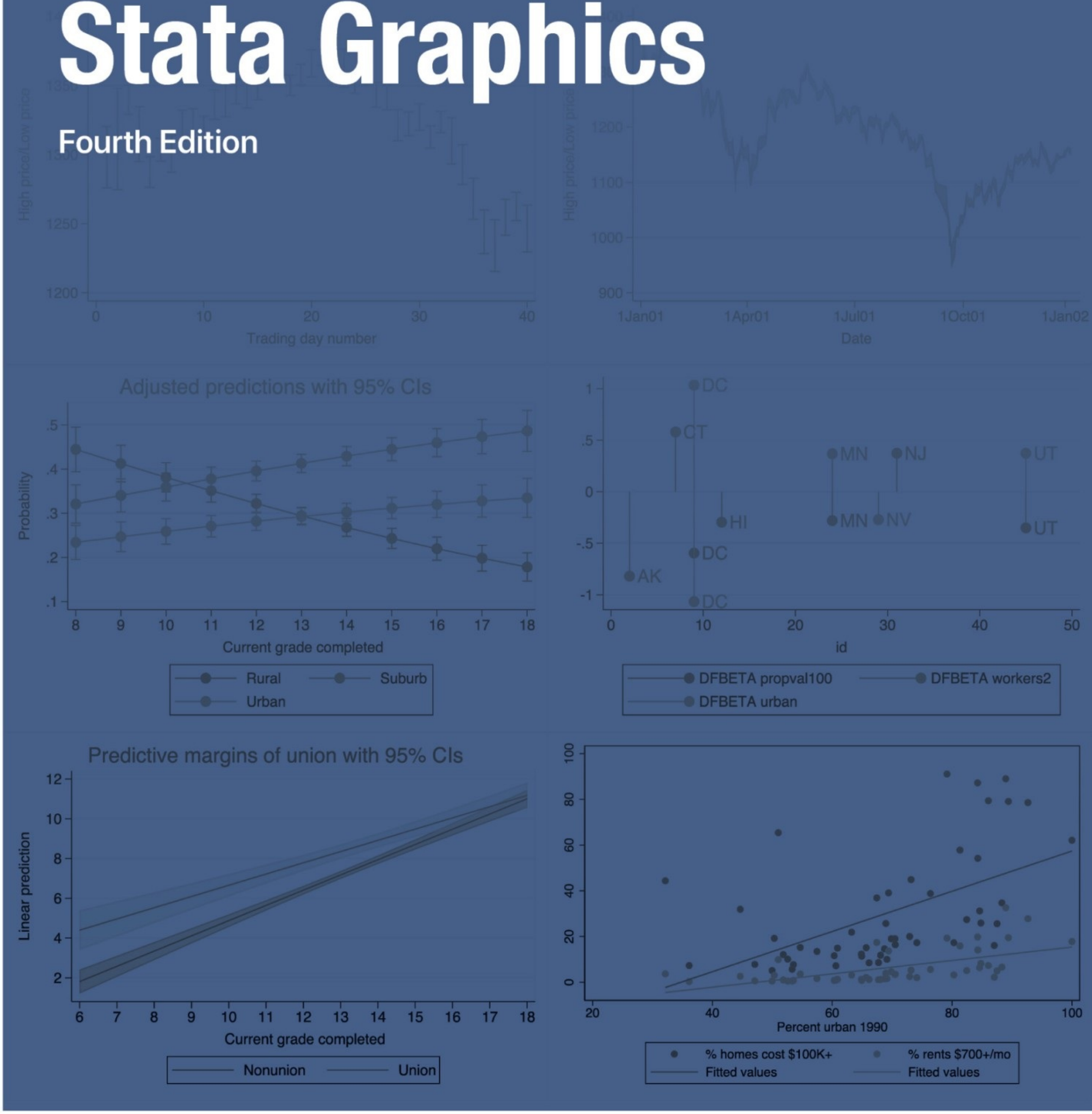
A Visual Guide to Stata Graphics

Fourth Edition

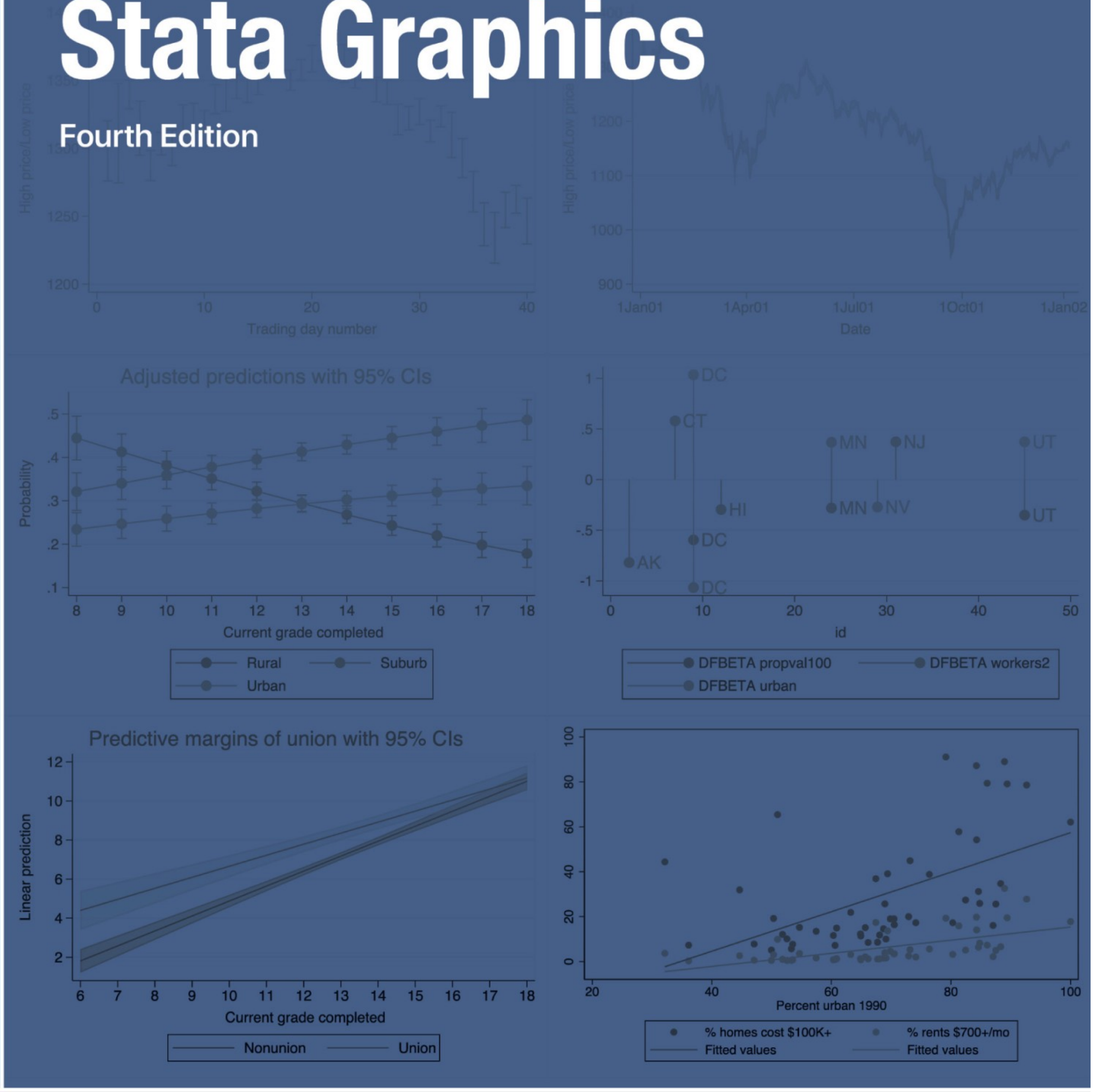


MICHAEL N. MITCHELL

Stata Press

A Visual Guide to Stata Graphics

Fourth Edition



MICHAEL N. MITCHELL

Stata Press

A Visual Guide to Stata Graphics

Fourth Edition

MICHAEL N. MITCHELL

A Stata Press Publication
StataCorp LLC
College Station, Texas

Copyright © 2004, 2008, 2012, 2022 by StataCorp LLC
All rights reserved. First edition 2004
Second edition 2008
Third edition 2012
Fourth edition 2022

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845

Typeset in LaTeX2e

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Print ISBN-10: 1-59718-365-2

Print ISBN-13: 978-1-59718-365-9

ePub ISBN-10: 1-59718-366-0

ePub ISBN-13: 978-1-59718-366-6

Library of Congress Control Number: 2021951939

No part of this book may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means—electronic, mechanical, photocopy, recording, or otherwise—without the prior written permission of StataCorp LLC.

Stata, stata, Stata Press, Mata, mata, and NetCourse are registered trademarks of StataCorp LLC.

Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations.

NetCourseNow is a trademark of StataCorp LLC.

LaTeX2e is a trademark of the American Mathematical Society.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Dedication

I dedicate this book to the teachers of the world. I have been fortunate to have been touched by many special teachers, and I will always be grateful for what they kindly gave to me. I thank (in order of appearance) Larry Grossman, Fred Perske, Rosemary Sheridan, Donald Butler, Jim Torcivia, Richard O'Connell, Linda Fidell, and Jim Sidanius. These teachers all left me gifts of knowledge and life lessons that help me every day. Even if they do not all remember me, I will always remember them.

Contents

Dedication

Preface to the Fourth Edition

Preface to the Third Edition

Preface to the Second Edition

Preface to the First Edition

Acknowledgments

1 Introduction

- 1.1 Online supplements
- 1.2 Using this book
- 1.3 Types of Stata graphs
- 1.4 Schemes
  - 1.4.1 Schemes included with Stata
  - 1.4.2 Community-contributed schemes
  - 1.4.3 Schemes included with this book
  - 1.4.4 Setting schemes
- 1.5 Options
- 1.6 Building graphs
- 1.7 Point-and-click interface

2 Twoway

- 2.1 Scatter
- 2.2 Fit
- 2.3 CI fit
- 2.4 Line
- 2.5 Area
- 2.6 Bar
- 2.7 Range
- 2.8 Distribution
- 2.9 Contour

- 2.10 Options
- 2.11 Overlaying

3 Matrix

- 3.1 Marker options
- 3.2 Axes
- 3.3 Matrix options
- 3.4 By

4 Bar

- 4.1 Y variables
- 4.2 Over
- 4.3 Bar gaps
- 4.4 Bar sorting
- 4.5 Cat axis
- 4.6 Legend and labels
- 4.7 Y axis
- 4.8 Lookofbar options
- 4.9 By

5 Box

- 5.1 Yvars and over
- 5.2 Box gaps
- 5.3 Box sorting
- 5.4 Cat axis
- 5.5 Legend
- 5.6 Y axis
- 5.7 Boxlook options
- 5.8 By

6 Dot

- 6.1 Yvars and over
- 6.2 Dot gaps
- 6.3 Dot sorting
- 6.4 Cat axis
- 6.5 Legend
- 6.6 Y axis
- 6.7 Dotlook options

  - 6.8 By

7 Pie

  - 7.1 Types of pie charts
  - 7.2 Sorting
  - 7.3 Colors and exploding
  - 7.4 Labels
  - 7.5 Legend
  - 7.6 By

8 Options

  - 8.1 Markers
  - 8.2 Marker labels
  - 8.3 Connecting
  - 8.4 Axis titles
  - 8.5 Axis labels
  - 8.6 Axis scales
  - 8.7 Axis selection
  - 8.8 By
  - 8.9 Legend
  - 8.10 Adding text
  - 8.11 Textboxes
  - 8.12 Text display

9 Standard options

  - 9.1 Titles
  - 9.2 Schemes
    - 9.2.1 Schemes included with Stata
    - 9.2.2 Community-contributed schemes
    - 9.2.3 Schemes included with this book
    - 9.2.4 Example #1: An overlaid scatterplot with fit lines
    - 9.2.5 Example #2: An overlaid scatterplot with fit lines and confidence region
    - 9.2.6 Customizing schemes
    - 9.2.7 Using the set scheme command
  - 9.3 Sizing graphs
    - 9.3.1 Sizing/resizing graphs with absolutely sized versus relatively sized units

- 9.4 Graph regions

10 Styles

- 10.1 Angle
- 10.2 Color
  - 10.2.1 Named colors
  - 10.2.2 Color intensity
  - 10.2.3 Color opacity
  - 10.2.4 Overlapping colors
  - 10.2.5 Specifying colors using RGB, CMYK, and HSV values
- 10.3 Clock position
- 10.4 Compass
- 10.5 Connect
- 10.6 Line pattern
- 10.7 Line width
- 10.8 Margin
- 10.9 Marker size
- 10.10 Orientation
- 10.11 Marker symbol
- 10.12 Text size

11 Appendix

- 11.1 Stat
- 11.2 Stat options
- 11.3 marginsplot
- 11.4 Save/Redisplay/Combine
- 11.5 Export
- 11.6 More examples
- 11.7 Common mistakes

Subject index

Preface to the Fourth Edition

When I was writing the first edition of this book, we all pictured a book printed in black and white. All the other books in the Stata Press catalog were in black and white. As the book was nearing completion, Stata Press found a printer who could print the book in full color. The book was nearly done—nearly done in black and white. I took a hard swallow, and we agreed that even though it would take extra time and rethinking parts of the book from scratch, the book should be in color. Seeing all the features that Stata has added for supporting colors, I find it hard to imagine this book any other way. This new edition goes *all in* on the features that Stata offers for displaying colors. In the third edition, the section on color styles had five examples—that section in this new edition includes over 50 examples [see Styles : Colors (section 10.2)]. Instead of trying to explain the look of colors at different intensities and opacities, I show you commands and graphs that illustrate different colors shown at differing intensities and using differing opacities. Further, I illustrate how these options interact when regions with different colors are overlaid atop each other. You can play with these examples to explore other combinations of colors/intensities/opacities, either alone or when overlapping one another.

In addition to the new coverage of colors, this new edition details the methods you can use for sizing objects, showing the three ways of sizing objects using *absolute units* (like points, inches, and centimeters) and the three ways you can size objects using *relative units* (such as using keywords like `large`, multipliers of the original size like `*2`, or sizes relative to the size of the graph, like `5rs`). Each of these units is illustrated in the context of sizing different elements, such as text [like titles, axis labels, marker labels, legends, and so on; see Styles : Textsize (section 10.12)]; markers [see Styles : Markersize (section 10.9)]; line widths [see Styles : Linewidth (section 10.7)]; and more. Each of those sections illustrates sizing of elements in isolation—additionally, Standard options : Sizing graphs (section 9.3) illustrates resizing the entire graph and the different results you obtain when individual elements are sized using relative units versus absolute units.

If you have used prior editions of the book, you may notice that this edition no longer includes a chapter on the Graph Editor and that the examples focus exclusively on the use of commands for creating graphs. This is not a commentary about the utility of the Graph Editor, but instead a reflection that this book was getting too large and that Stata has a growing library of video tutorials that *interactively show* how to create and modify graphs via the Stata interface. In section 1.7, I describe the utility of the interactive point-and-click interface for creating and modifying graphs and suggest videos I think illustrate key features.

The overall look of this book is dramatically different from the prior edition. The prior editions periodically changed the schemes to introduce novelty and pizzazz and to underscore how powerful schemes are for controlling the entire look of your graph. This new edition uses one common scheme and changes the scheme only when there is a rationale for choosing one scheme over another. With schemes in mind, the heart of section Standard options : Schemes (section 9.2) shows three different kinds of graphs, one at a time, illustrating the look of that graph using selected schemes that ship with Stata, schemes included with this book, and several schemes from the worldwide Stata community.

Writing this fourth edition book was a great pleasure, especially for the respite it gave during such difficult and turbulent times. I deeply hope that this book finds you happy, healthy, and—most of all—safe.

Ventura, California December 2021

Preface to the Third Edition

This third edition updates the second edition of this book, reflecting new features available in Stata version 12. Since version 10, Stata has added several new graphical features, including a command for creating contour plots, options that give you greater control over the display of text, and the ability to create graphs from the results of the `margins` command. Additional sections have been added to this third edition that illustrate these new features.

A new section has been added that illustrates the use of the `twoway contour` command; see Twoway : Contour (section 2.9). You can see Options : Text Display (section 8.12) for information about how to specify symbols, subscripts, and superscripts, as well as how to display text in bold or italics; this section also describes how you can display text using different fonts. A new section has also been added that describes how you can customize graphs created using the `marginsplot` command; see Appendix : Marginsplot (section 11.3).

This third edition also includes minor updates here and there to bring the text up to date for use with Stata version 12.

Simi Valley, California December 2011

Preface to the Second Edition

I cannot believe that it has been over three years since the release of the first edition of this book. A lot has changed since then, and that includes the way that Stata graphics have evolved. Although the core features remain the same, there have been many enhancements and more features added, the most notable being the addition of the interactive point-and-click Stata Graph Editor.

The second edition of this book has been thoroughly revised to address these new features, especially the Graph Editor. This edition has an entire chapter devoted to the use of the Graph Editor. Also, almost every example in this book has been augmented to include descriptions of how the Graph Editor can be used to create the customizations being illustrated via commands.

The Stata Graph Editor and Stata `graph` commands offer powerful tools for customizing your graphs, and I hope that the coverage of both side by side helps you to use each to their fullest capacity. To emphasize this point, I wrote a section that describes certain areas where I feel that commands are especially superior to the Graph Editor, and areas where I feel the Graph Editor is especially superior to commands. Although I still feel that commands provide a primary mode of creating graphs, you need to use the Graph Editor for only a short amount of time to see what a smart and powerful tool it is. Whereas commands offer the power of repeatability, the Graph Editor provides a nimble interface that permits you to tangibly modify graphs like a potter directly handling clay. I hope that this book helps you to integrate the effective use of both of these tools into your graph-making toolkit.

As with the first edition, updating this second edition has been both a challenge and a delight. I have endeavored to make this book a tool that you would find friendly, logical, intuitive, and above all, useful. I really hope you like it!

Simi Valley, California April 2008

Preface to the First Edition

It is obvious to say that graphics are a visual medium for communication. This book takes a visual approach to help you learn about how to use Stata graphics. While you can read this book in a linear fashion or use the table of contents to find what you are seeking, it is designed to be “thumbed through” and visually scanned. For example, the right margin of each right page has what I call a *Visual Table of Contents* to guide you through the chapters and sections of the book. Generally, each page has three graphs on it, allowing you to see and compare as many as six graphs at a time on facing pages. For a given graph, you can see the command that produced it, and next to each graph is some commentary. But don’t feel compelled to read the commentary; often, it may be sufficient just to see the graph and the command that made it.

This is an informal book and is written in an informal style. As I write this, I picture myself sitting at the computer with you, and I am showing you examples that illustrate how to use Stata graphics. The comments are written very much as if we were sitting down together and I had a couple of points to make about the graph that I thought you might find useful. Sometimes, the comments might seem obvious, but because I am not there to hear your questions, I hope it is comforting to have the obvious stated just in case there was a bit of doubt.

While this book does not spend much time discussing the syntax of the graph commands (because you will be able to infer the rules for yourself after seeing a number of examples), the Intro : Options (section 1.5) section discusses some of the unique ways that options are used in Stata graph commands and compares them with the way that options are used in other Stata commands.

I strived to find a balance to make this book comprehensive but not overwhelming. As a result, I have omitted some options I thought would be seldom used. So, just because a feature is not illustrated in this book, this does not mean that Stata cannot do that task, and I would refer to [G-2] **graph** for more details. I try to include frequent cross-references to [G-

2] **graph**; for example, see also [G-3] ***axis_options***. I view this book as a complement to the *Stata Graphics Reference Manual*, and I hope that these cross-references will help you use these two books in a complementary manner. Note that, whenever you see references to [G-2] **xyz**, you can either find “xyz” in the *Stata Graphics Reference Manual* or type `whelp xyz` within Stata. The manual and the help have the same information, although the help may be more up to date and allows hyperlinking to related topics.

Each chapter is broken into a number of sections showing different features and options for the particular kind of graph being discussed in the chapter. The examples illustrate how these options or features can be used, focusing on examples that isolate these features so you are not distracted by irrelevant aspects of the Stata command or graph. While this approach improves the clarity of presentation, it does sacrifice some realism because graphs frequently have many options used together. To address this, there is a section addressing strategies for building up more complicated graphs, Intro : Building graphs (section 1.6), and a section giving tips on creating more complicated graphs, Appendix : More examples (section 11.6). These sections are geared to help you see how you can combine options to make more complex and feature-rich graphs.

While this book is printed in color, this does not mean that it ignores how to create monochrome (black & white) graphs. Some of the examples are shown using monochrome graphs illustrating how you can vary colors using multiple shades of gray and how you can vary other attributes, such as marker symbol and size, line width, and pattern, and so forth. I have tried to show options that would appeal to those creating color or monochrome graphs.

The graphs in this book were created using a set of schemes specifically created for this book. Despite differences in their appearance, all the schemes increase the size of textual and other elements in the graphs (for example, titles) to make them more readable, given the small size of the graphs in this book. You can see more about the schemes in Intro : Schemes (section 1.4) and how to obtain them in Introduction : Online supplements (section 1.1). While one purpose of the different schemes is to aid in your visual enjoyment of the book, they are also used to illustrate the utility of schemes for setting up the look and default settings for your graphs. See Introduction : Online supplements (section 1.1) for information about how you can obtain these schemes.

Stata has many graph commands for producing special-purpose statistical graphs. Examples include graphs for examining the distributions of variables (for example, `kdensity`, `pnorm`, or `gladder`), regression diagnostic plots (for example, `rvfplot` or `lvr2plot`), survival plots (for example, `sts` or `ltable`), time series plots (for example, `ac` or `pac`), and ROC plots (for example, `roctab` or `lsens`). To cover these graphs in enough detail to add something worthwhile would have expanded the scope and size of this book and detracted from its utility. Instead, I have included a section, Appendix : Stat graphs (section 11.1), that illustrates a number of these kinds of graphs to help you see the kinds of graphs these commands create. This is followed by Appendix : Stat graph options (section 11.2), which illustrates how you can customize these kinds of graphs using the options illustrated in this book.

If I may close on a more personal note, writing this book has been very rewarding and exciting. While writing, I kept thinking about the kind of book you would want to help you take full advantage of the powerful, but surprisingly easy to use, features of Stata graphics. I hope you like it!

Simi Valley, California February 2004

Acknowledgments

Every book that I write with Stata Press is a very collaborative endeavor. I have benefited tremendously from the collaboration on all of my books, but none more than this graphics book. This book really owes a debt to Jeff Pitblado and Lisa Gilmore of the Stata Press team for creating the LaTeX wizardry that makes the unique layout of this book possible. Additionally, they have created and adapted tools to make the electronic editions possible.

In this fourth edition, I received so much input and inspiration from Kristin MacDonald and fantastic technical editing from Derek Wagner. I am so grateful for their suggestions and insights!

I also thank David Culwell for his very helpful, detailed, and thorough editing. Finally, I wish to express my deep thanks to Eric Hubbard for his delightful cover design.

Chapter 1
Introduction

This chapter begins by briefly telling you about how to access the datasets and schemes used in this book, so you can replicate and extend any of the examples for yourself. The next section gives you some tips about using this book, followed by short overview of the different kinds of Stata graphs that will be examined in this book. Next, I provide an overview of schemes and how they can be used to obtain different looks for graphs. The fourth section illustrates the structure of options in Stata graph commands. In a sense, the third, fourth, and fifth sections of this chapter are a thumbnail preview of the entire book, showing the types of graphs covered, how you can control their overall look, and the general structure of options used within those graphs. The next section is about the process of creating graphs, and the final section provides information about the interactive point and click interface for creating and editing graphs

1.1 Online supplements

I encourage you to download the data and schemes associated with this book. That will allow you to replicate, and extend, the examples shown in this book. You can quickly download all the datasets, schemes, and programs used in this book with the following `net` commands.

```
. net from https://www.stata-press.com/data/vgsg4/
. net get vgsg4
. net install vgsg4
```

The `net from` command connects you to the resources associated with this book. The `net get` command will download the datasets into your current working directory.[1] The `net install` command will install the schemes used in this book as well as the programs used in this book (for example, `vgcolormap`).[2]

You can visit the webpage for the book at

https://www.stata-press.com/books/visual-guide-to-stata-graphics/

This site will have additional information about the book, any updates on obtaining the latest scheme files illustrated in the book, and an *Errata* showing any errors that have been found.

Each graph shows the dataset used prior to creating the graph, for example,

Uses allstates.dta

This statement indicates that you will want to read the dataset `allstates.dta` into memory before issuing the graph command. The default scheme used for the book is `vg_s2cx`. If a different scheme is used, it will be specified via the `scheme()` option. For more information about schemes, see Intro : Schemes (section 1.4) and Standard options : Schemes (section 9.2).

---

[1] You may want to store these datasets in a specific folder. In that case, prior to the `net get` command, you may want to make a folder for the datasets and then use the `cd` command to make that your current working directory.

[2] If you have installed the schemes/programs previously, you will need to add the `replace` option—that is, `net install vgsg4, replace`.

1.2 Using this book

I hope that you are eager to start reading this book but will take just a couple of minutes to read this section to get some suggestions that will make the book more useful to you. There are many ways you might read this book, but perhaps I can suggest some tips:

- Read this chapter before reading the other chapters, as it provides key information that will make the rest of the book more understandable.
- Although you might read a traditional book cover to cover, this book has been written so that the chapters stand on their own. You should feel free to dive into any chapter or section of any chapter.
- Sometimes you might find it useful to visually scan the graphs rather than to read. I think this is a good way to familiarize yourself with the kinds of features available in Stata graphs. If a certain feature catches your eye, you can stop and see the command that made the graph and even read the text explaining the command.
- Likewise, you might scan a chapter just by looking at the graphs and the part of the command in red, which is the part of the command highlighted in that graph. For example, scanning the chapter on bar charts in this way would quickly familiarize you with the kinds of features available for bar graphs and would show you how to obtain those features.

The right margin contains what I call the *Visual Table of Contents*. It is a useful tool for quickly finding the information you seek. I frequently use the *Visual Table of Contents* to cross-reference information within the book. By design, Stata graphs share many common features. For example, you use the same kinds of options to control a legend across different types of graphs. It would be repetitive to go into detail about a legend for bar charts, box plots, and so on. Within each kind of graph, a legend is briefly described and illustrated, but the details are described in the *Options* chapter in the section titled *Legend*. This is cross-referenced in the book by saying something like “for more details, see Options : Legend (section 8.9)”, indicating that you should look to the *Visual Table of*

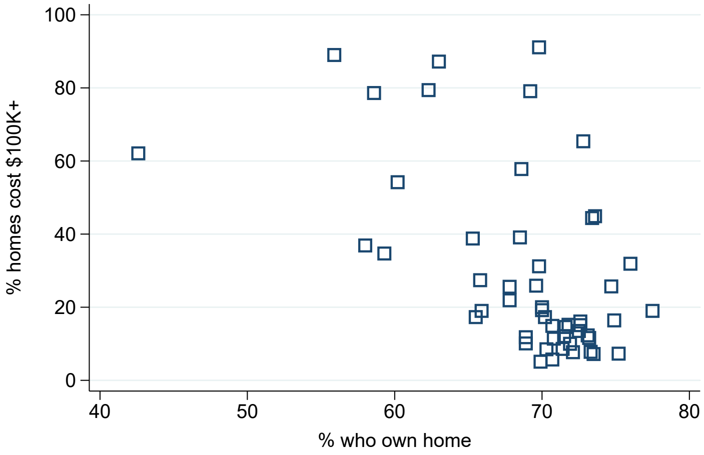
Contents and thumb to the *Options* chapter and then to the *Legend* section, which begins in section 8.9.

Sometimes it may take an extra cross-reference to get the information you need. Say that you want to make the y-axis title large for a bar chart by using the `ytitle()` option, so you first consult Bar : Y-axis (section 4.7). This gives you some information about using `ytitle()`, but then that section refers you to Options : Axis titles (section 8.4), where more details about axis titles are described. This section then refers you to Options : Textboxes (section 8.11) for more complete details about options to control the display of text. That section shows more details but then refers to Styles : Textsize (section 10.12), where all the possible text sizes are described. I know this sounds like a lot of jumping around, but I hope that it feels more like drilling down for more detail, that you feel you are in control of the level of detail that you want, and that the *Visual Table of Contents* eases the process of getting the additional details.

Most pages of this book have three graphs per page, with each graph being composed of the graph itself, the command that produced it, and some descriptive text. An example is shown below, followed by some points to note.

```
graph twoway scatter propval100 ownhome, msymbol(Sh)
```

Here we use the `msymbol()` (marker symbol) option to make the symbols large hollow squares; see Options : Markers (section 8.1) for more details. The `graph twoway` portion of the command is optional. *Uses allstates.dta*



- The command itself is displayed in a `typewriter font`, and the salient part of the command (that is, `msymbol(Sh)`) is in `this color`—both in the command and when referenced in the descriptive text.
- When commands or parts of commands are given in the descriptive text (for example, `graph twoway`), they are displayed in the `typewriter font`.
- Many of the descriptions contain cross-references, for example, Options : Markers (section 8.1), which means to flip to the *Options* chapter and then to the section *Markers*. Equivalently, go to section 8.1.
- The names of some options are shorthand for two or more words that are sometimes explained; for instance, “we use the `msymbol()` (marker symbol) option to make …”.
- The descriptive text always concludes by telling you the name of the data file in memory when making the graph. Here the data file was `allstates.dta`.

If you want your graphs to look like the ones in the book, you can display them using the same schemes. See Introduction : Online supplements (section 1.1) for information about how to download the schemes used in this book. Once you have downloaded the schemes, you can then type the following commands in the Stata Command window:

```
. set scheme vg_s2cx
. use allstates
. graph twoway scatter propval100 ownhome, msymbol(Sh)
```

After you issue the `set scheme vg_s2cx` command, subsequent graph commands will show graphs with the `vg_s2cx` scheme. You could also add the `scheme(vg_s2cx)` option to the graph command to specify that the scheme be used just for that graph; for example,

```
. graph twoway scatter propval100 ownhome, msymbol(Sh) scheme(vg_s2cx)
```

Generally, all commands and options are provided in their complete form. Commands and options are usually not abbreviated. However, for purposes of typing, you may want to use abbreviations. The previous example could have been abbreviated to

```
. gr tw sc propval100 ownhome, m(Sh)
```

The `gr` could have been omitted, leaving

```
. tw sc propval100 ownhome, m(Sh)
```

The `tw` also could have been omitted, leaving

```
. sc propval100 ownhome, m(Sh)
```

For guidance on appropriate abbreviations, consult `help graph`.

This book has been written based on the features available in Stata version 17. In the future, Stata may evolve to make the behavior of some of these commands change. If this happens, you can use the `version` command to make Stata run the graph commands as though they were run under version 17. For example, if you were running Stata version 18.0 but wanted a graph command to run as though you were running Stata 17, you could type

```
. version 17: graph twoway scatter propval100 ownhome
```

and the command would be executed as if you were running version 17. Or, perhaps you want a command to run as it did under Stata 16.1, you would then type

```
. version 16.1: graph twoway scatter propval100 ownhome
```

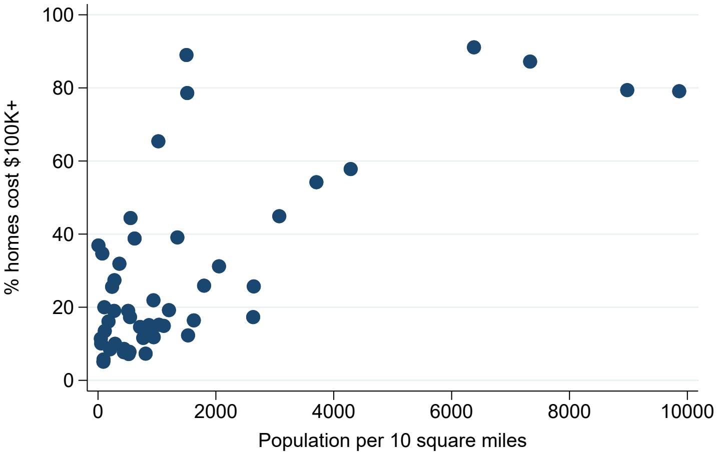
Finally, I would like to emphasize that the goal of this book is to help you learn and use the Stata graph commands for the purposes of creating graphs in Stata. I assume that you know the kind of graph you want to create and that you are turning to this book for advice on how to make that graph. I don’t provide guidance on how to select the right kind of graph for visualizing your data or the merits of one graphical method over another. For such guidance, I would refer readers to books such as *The Visual Display of Quantitative Information, Second Edition* by Edward R. Tufte and *Visualizing Data* by William S. Cleveland. Additionally, if you are creating a graph as part of a manuscript to be submitted for publication, I recommend consulting the author guidelines from the publisher as well as looking at graphs that have been recently been published in the journal for guidelines for creating your graphs.

1.3 Types of Stata graphs

Stata has a wide variety of graph types. This section introduces the types of graphs Stata produces, and it covers twoway plots (including scatterplots, line plots, fit plots, fit plots with confidence intervals, area plots, bar plots, range plots, and distribution plots), scatterplot matrices, bar charts, box plots, dot plots, and pie charts. Let's begin by exploring the variety of twoway plots that can be created with `graph twoway`. For this introduction, they are combined into six families of related plots: scatterplots and fit plots, line plots, area plots, bar plots, range plots, and distribution plots. Now let's turn to scatterplots and fit plots.

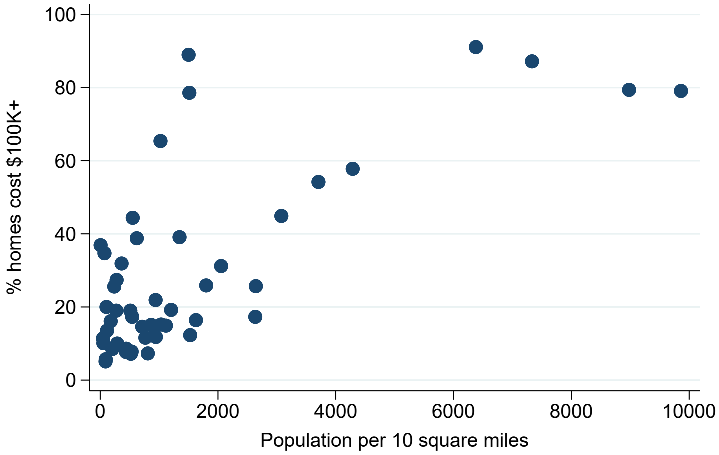
```
graph twoway scatter propval100 popden
```

Here is a basic scatterplot. The variable `propval100` is placed on the y axis, and `popden` is placed on the x axis. See Twoway : Scatter (section 2.1) for more details about these kinds of plots. *Uses allstates.dta*



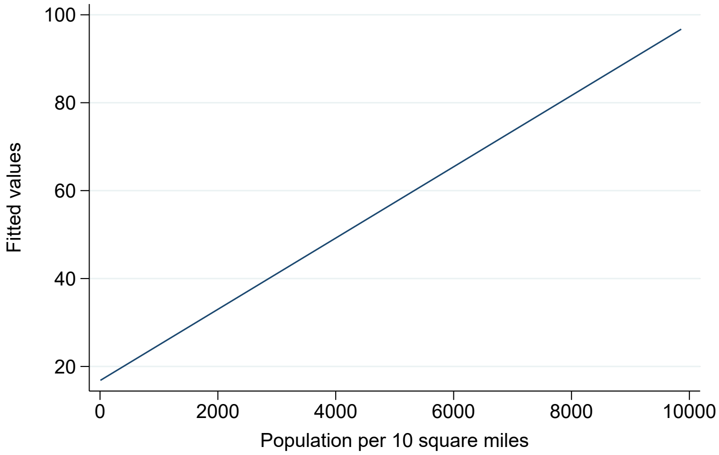
```
twoway scatter propval100 popden
```

We can start the previous command with just `twoway`, and Stata understands that this is shorthand for `graph twoway`. *Uses allstates.dta*



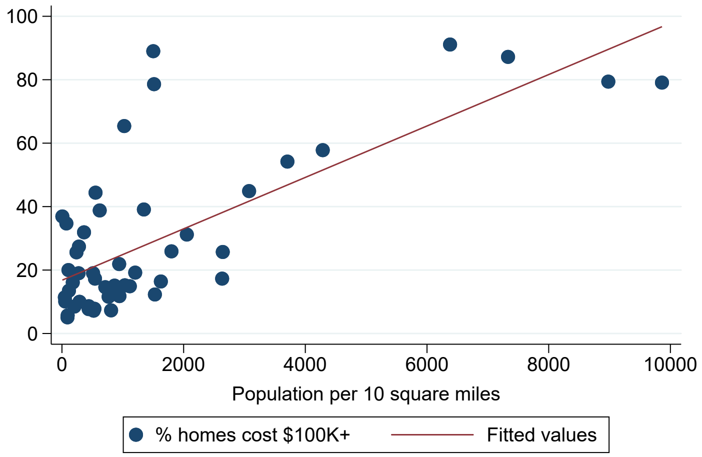
```
twoway lfit propval100 popden
```

We now make a linear fit (`lfit`) line predicting `propval100` from `popden`. See Twoway : Fit (section 2.2) for more information about these kinds of plots. *Uses allstates.dta*



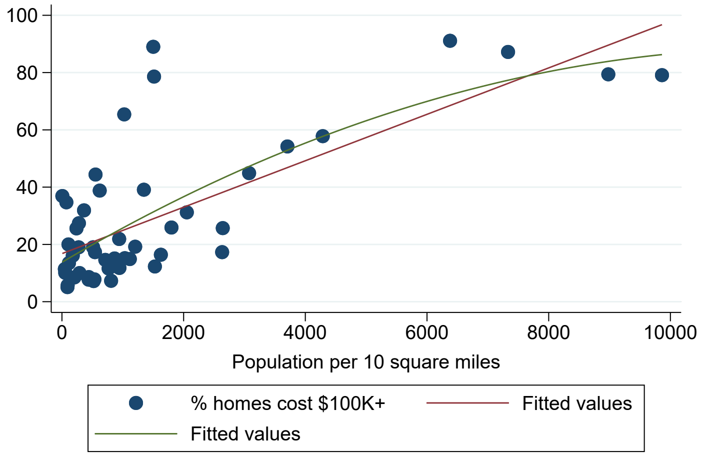
```
twoway (scatter propval100 popden) (lfit propval100 popden)
```

Stata allows us to overlay `twoway` graphs. In this example, we make a classic plot showing a scatterplot overlaid with a fit line by using the `scatter` and `lfit` commands. For more details about overlaying graphs, see Twoway : Overlaying (section 2.11). *Uses allstates.dta*



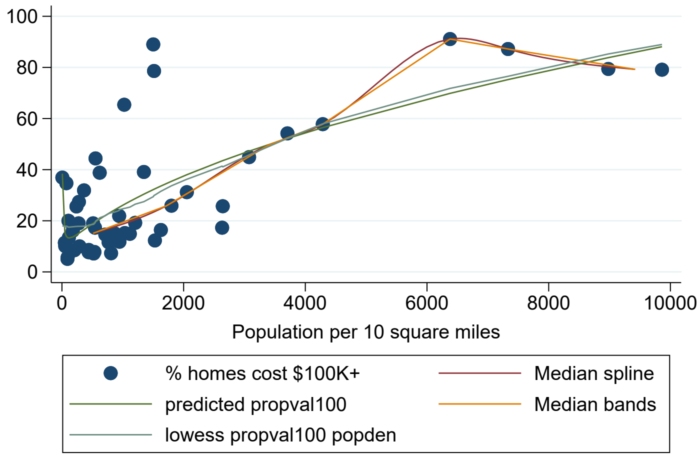
```
twoway (scatter propval100 popden) (lfit propval100 popden)
(qfit propval100 popden)
```

The ability to combine `twoway` plots is not limited to overlaying just two plots; we can overlay multiple plots. Here we overlay a scatterplot (`scatter`) with a linear fit (`lfit`) line and a quadratic fit (`qfit`) line. *Uses allstates.dta*



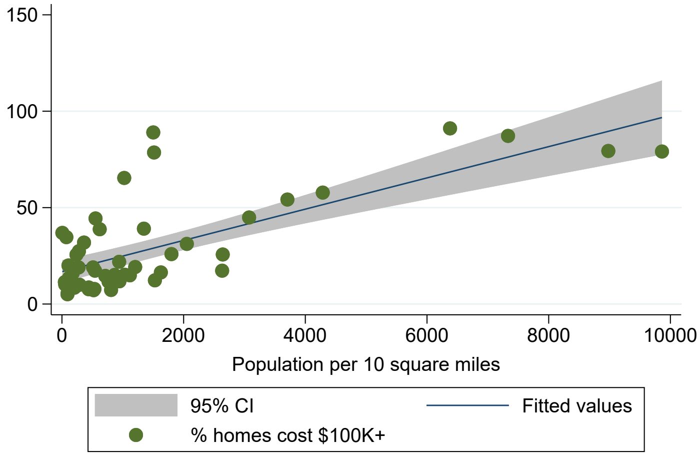
```
twoway (scatter propval100 popden) (mspline propval100
popden) (fpfit propval100 popden) (mband propval100 popden)
(lowess propval100 popden)
```

Stata has other kinds of fit methods in addition to linear and quadratic fits. This example includes a median spline (`mspline`), fractional polynomial fit (`fpfit`), median band (`mband`), and lowess (`lowess`). For more details, see Twoway : Fit (section 2.2). *Uses allstates.dta*



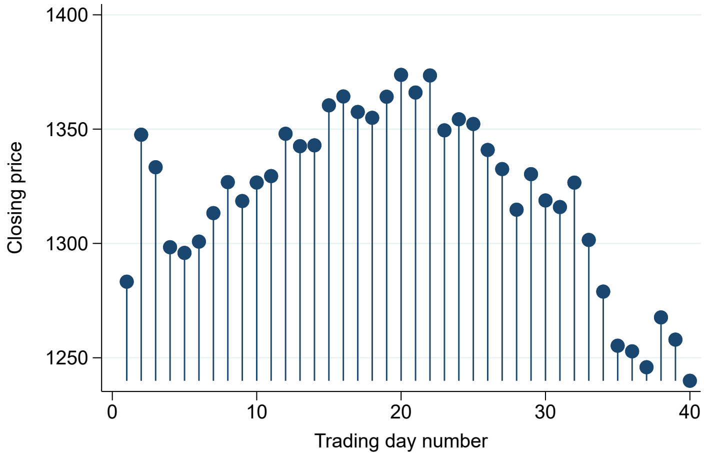
```
twoway (lfitci propval100 popden) (scatter propval100 popden)
```

In addition to being able to plot a fit line, we can plot a linear fit line with a confidence interval by using the `lfitci` command. We also overlay the linear fit and confidence interval with a scatterplot. See Twoway : CI fit (section 2.3) for more information about fit lines with confidence intervals. *Uses allstates.dta*



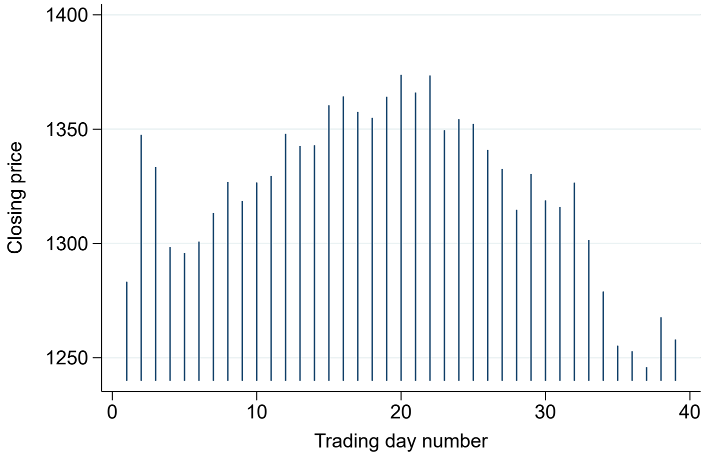
```
twoway dropline close tradeday
```

This `dropline` graph shows the closing prices of the S&P 500 by trading day for the first 40 days of 2001. A `dropline` graph is like a `scatter`plot because each data point is shown with a marker, but a dropline for each marker is shown as well. For more details, see Twoway : Scatter (section 2.1). *Uses spjanfeb2001.dta*



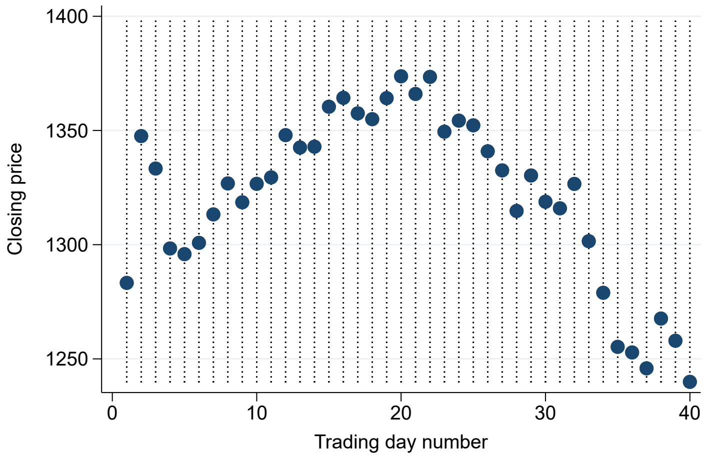
```
twoway spike close tradeday
```

Here we use a `spike` plot to show the same graph as the previous one. It is like the `dropline` plot, but no markers are put on the top. For more details, see **Twoway : Scatter** (section 2.1). *Uses spjanfeb2001.dta*



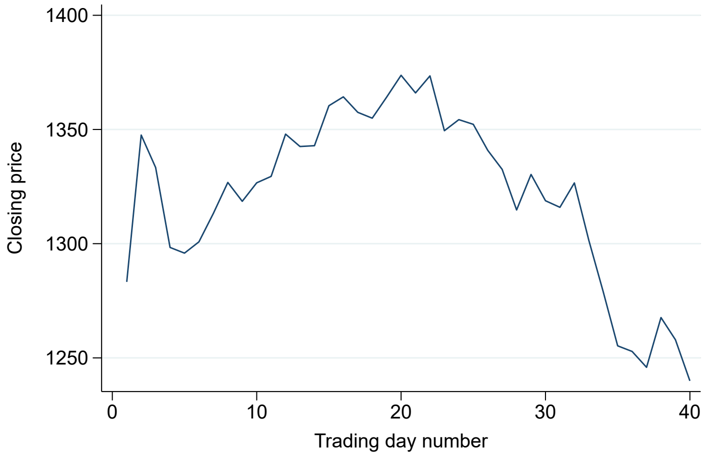
```
twoway dot close tradeday
```

The `dot` plot, like the `scatter`plot, shows markers for each data point but also adds a dotted line for each of the x values. For more details, see **Twoway : Scatter** (section 2.1). *Uses spjanfeb2001.dta*



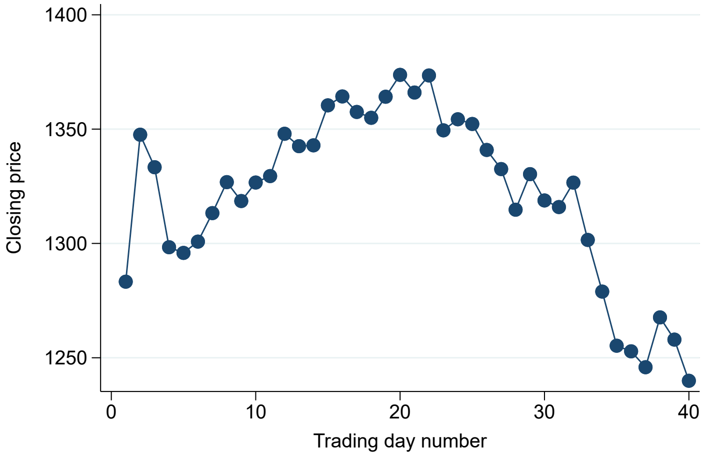
```
twoway line close tradeday, sort
```

We use the `line` command in this example to make a simple line graph. See Twoway : Line (section 2.4) for more details about line graphs. *Uses spjanfeb2001.dta*

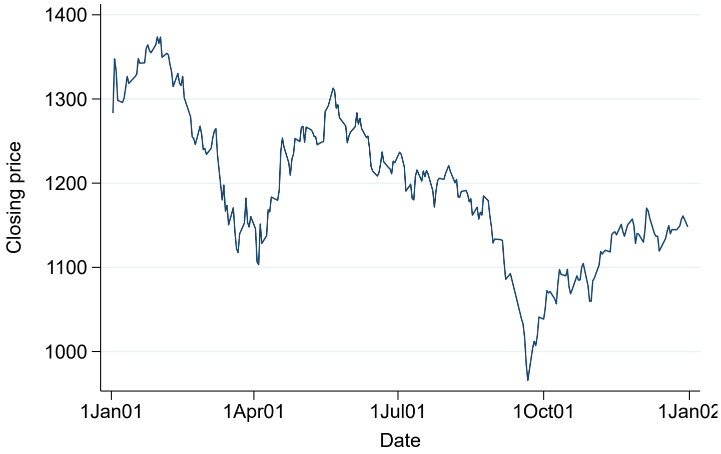


```
twoway connected close tradeday, sort
```

This `twoway connected` graph is similar to the `twoway line` graph, except that a symbol is shown for each data point. For more information, see Twoway : Line (section 2.4). *Uses spjanfeb2001.dta*



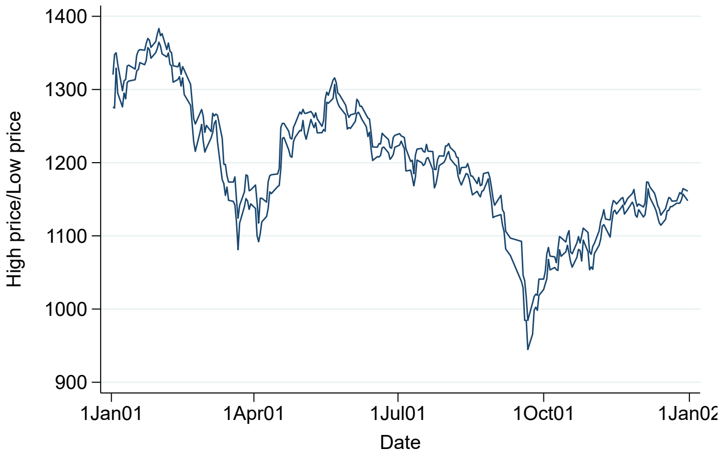
```
twoway tsline close, sort
```



The `tsline` (time-series line) command makes a line graph where the x variable is a date variable that has been previously declared by using `tsset`; see `help tsset`. This example shows the closing price of the S&P 500 by trading date. For more information, see Twoway : Line (section 2.4). *Uses sp2001ts.dta*

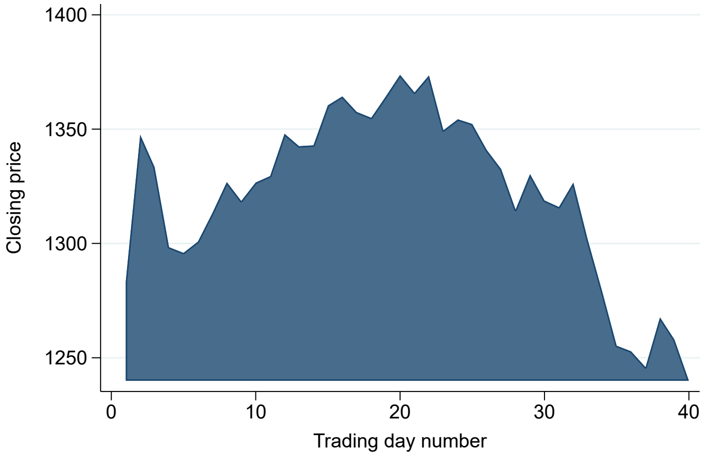
```
twoway tsrline high low, sort
```

The `tsrline` (time-series range line) command makes a line graph showing the high and low prices of the S&P 500 by trading date. For more information, see Twoway : Line (section 2.4). *Uses sp2001ts.dta*



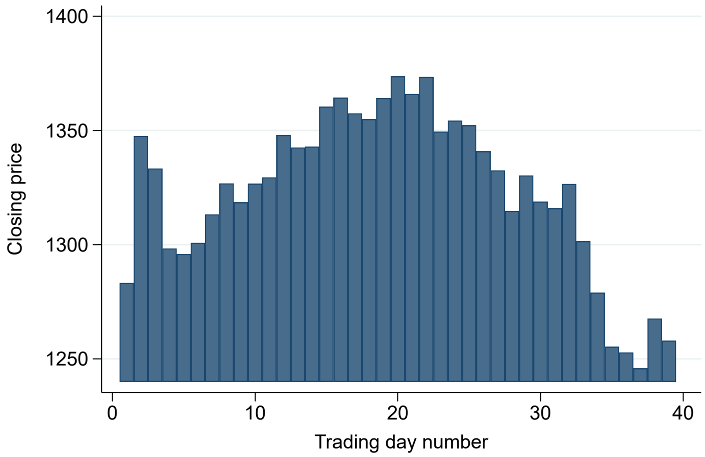
```
twoway area close tradeday, sort
```

An `area` plot is similar to a `line` plot, but the area under the line is shaded. See Twoway : Area (section 2.5) for more information about area plots. *Uses spjanfeb2001.dta*

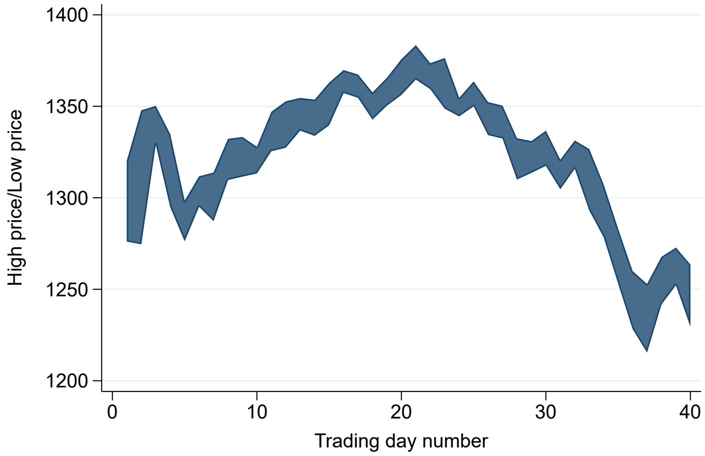


```
twoway bar close tradeday
```

Here is an example of a `twoway bar` plot. For each x value, a bar is shown corresponding to the height of the y variable. This command shows a continuous x variable as compared with the `graph bar` command, which would be useful when you have a categorical x variable. See Twoway : Bar (section 2.6) for more details about bar plots. *Uses spjanfeb2001.dta*



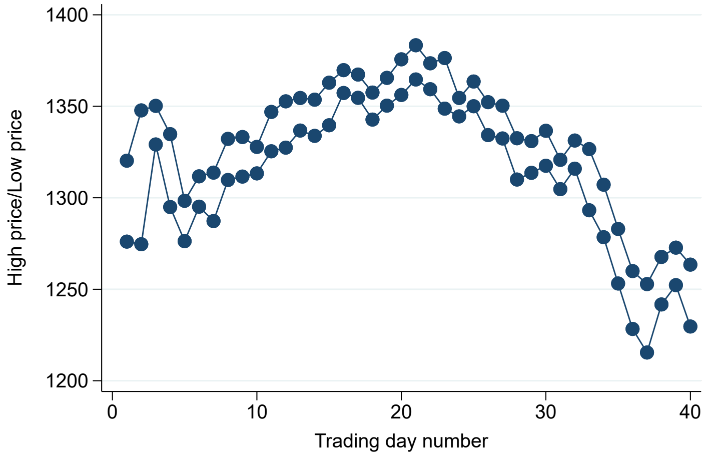
```
twoway rarea high low tradeday, sort
```



This example illustrates the use of `rarea` (range area) to graph the high and low prices with the area filled. If you used `rline` (range line), the area would not be filled. See Twoway : Range (section 2.7) for more details. *Uses spjanfeb2001.dta*

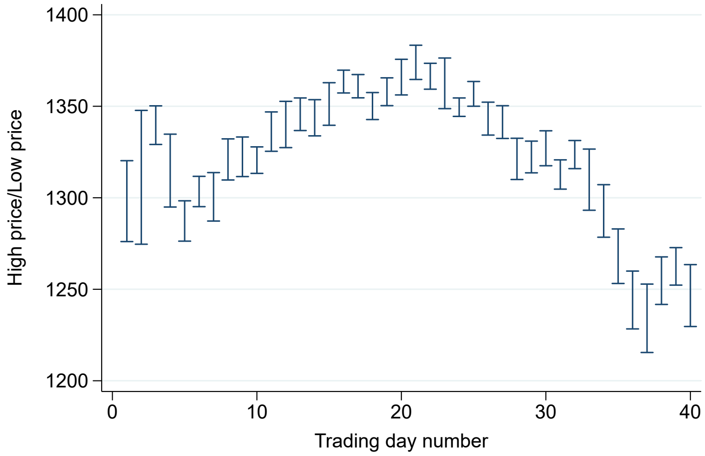
```
twoway rconnected high low tradeday, sort
```

The `rconnected` (range connected) command makes a graph similar to the previous one, except that a marker is shown at each value of the x variable and the area between is not filled. If you instead used `rscatter` (range scatter), the points would not be connected. See Twoway : Range (section 2.7) for more details. *Uses spjanfeb2001.dta*



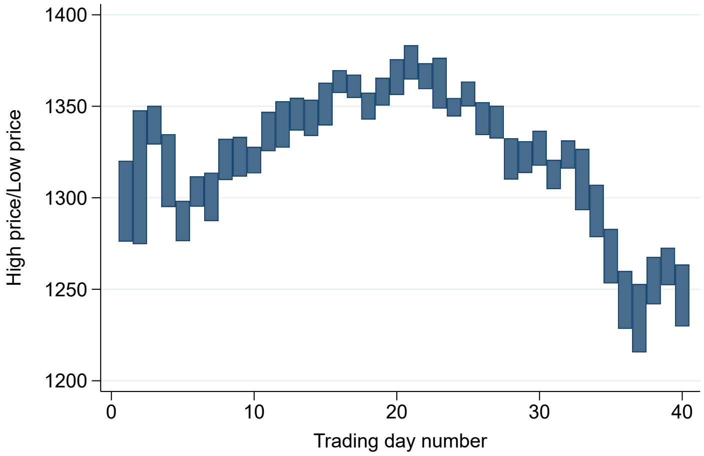
```
twoway rcap high low tradeday, sort
```

Here we use `rcap` (range cap) to graph the high and low prices with a spike and a cap at each value of the x variable. If you used `rspike` instead, spikes would be displayed but not caps. If you used `rcapsym`, the caps would be symbols that could be modified. See Twoway : Range (section 2.7) for more details. *Uses spjanfeb2001.dta*

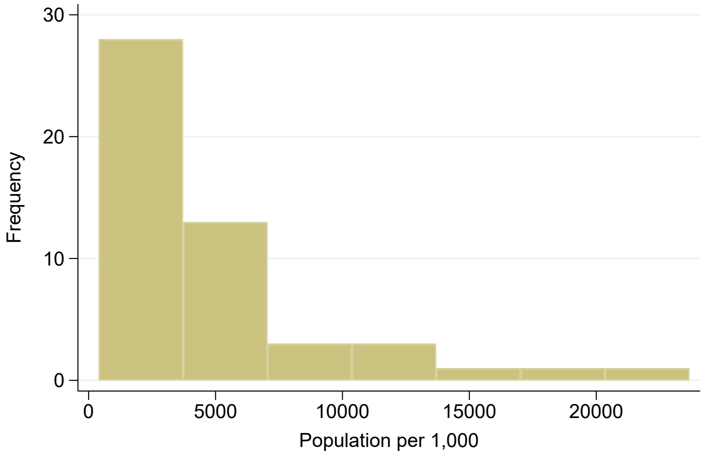


```
twoway rbar high low tradeday, sort
```

The `rbar` command graphs the high and low prices with bars at each value of the x variable. See Twoway : Range (section 2.7) for more details. *Uses spjanfeb2001.dta*



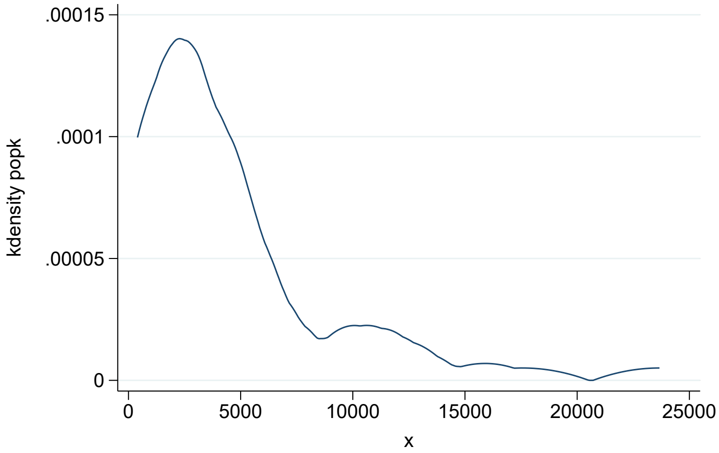
```
twoway histogram popk, freq
```



The `twoway histogram` command shows the distribution of one variable. It is often useful when overlaid with other twoway plots; otherwise, the `histogram` command would be preferable. See Twoway : Distribution (section 2.8) for more details. *Uses allstates.dta*

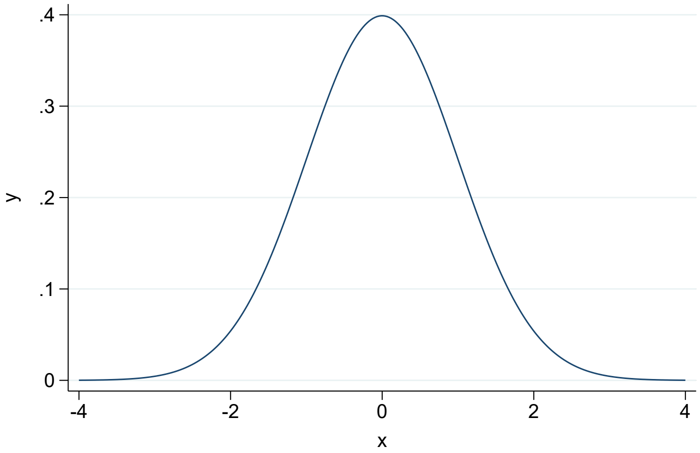
```
twoway kdensity popk
```

The `twoway kdensity` command shows a kernel density plot and is useful for examining the distribution of one variable. It can be overlaid with other twoway plots; otherwise, the `kdensity` command would be preferable. See Twoway : Distribution (section 2.8) for more details. *Uses allstates.dta*



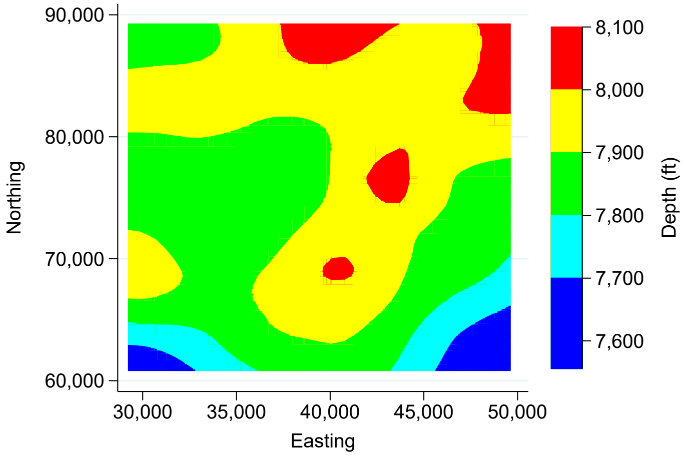
```
twoway function y=normalden(x), range(-4 4)
```

The `twoway function` command allows an arbitrary function to be drawn over a range of specified values. See Twoway : Distribution (section 2.8) for more details. *Uses allstates.dta*



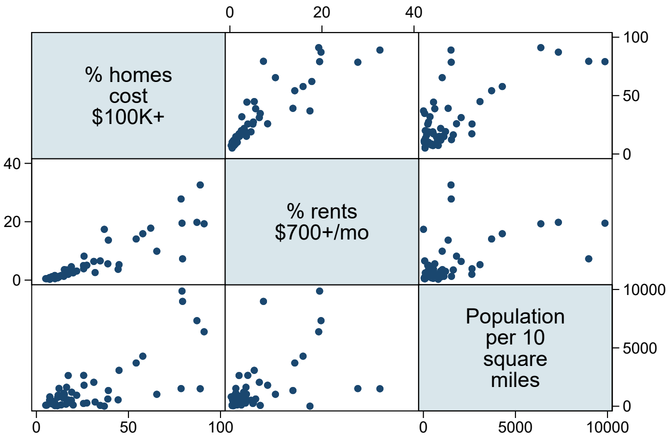
```
twoway contour depth northing easting
```

The `twoway contour` command creates contour plots representing three-dimensional data in two dimensions. See Twoway : Contour (section 2.9) for more details. *Uses sandstone.dta*



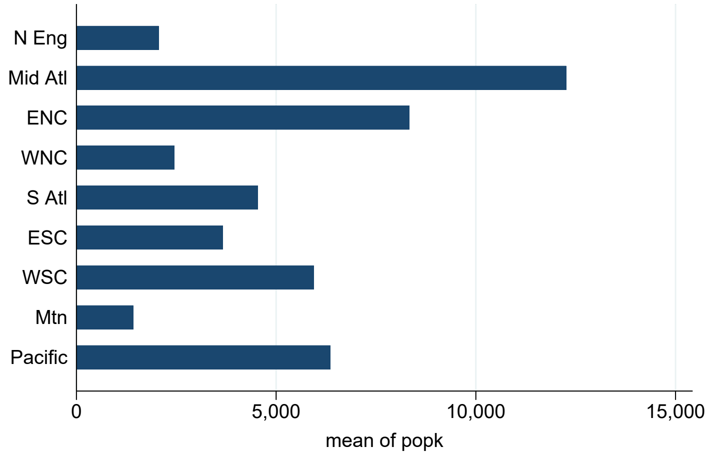
```
graph matrix propval100 rent700 popden
```

The `graph matrix` command shows a scatterplot matrix. See Matrix (section [3](#)) for more details. *Uses allstates.dta*

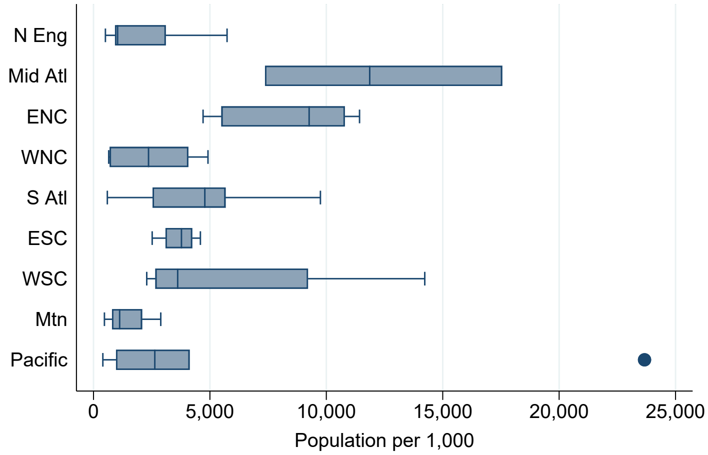


```
graph hbar popk, over(division)
```

This example shows how the `graph hbar` (horizontal bar) command is often used to show the values of a continuous variable broken down by one or more categorical variables. `graph hbar` is merely a rotated version of `graph bar`. See Bar (section [4](#)) for more details. *Uses allstates.dta*



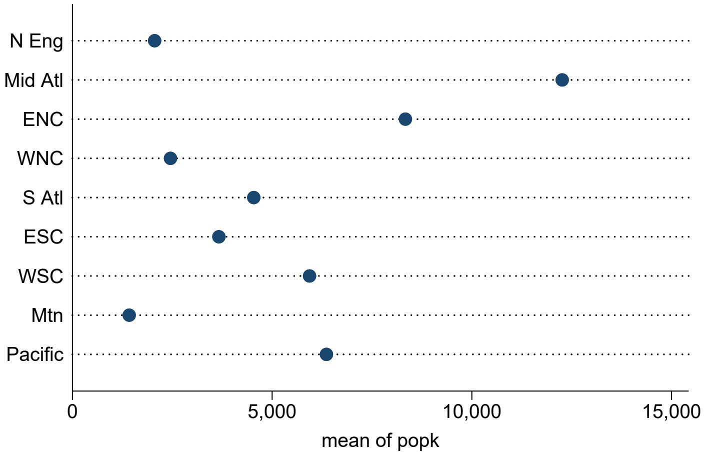
```
graph hbox popk, over(division)
```



Here is the previous graph as a box plot by using the `graph hbox` (horizontal box) command, which is commonly used for showing the distribution of one or more continuous variables, broken down by one or more categorical variables. `graph hbox` is merely a rotated version of `graph box`. See Box (section 5) for more details. *Uses allstates.dta*

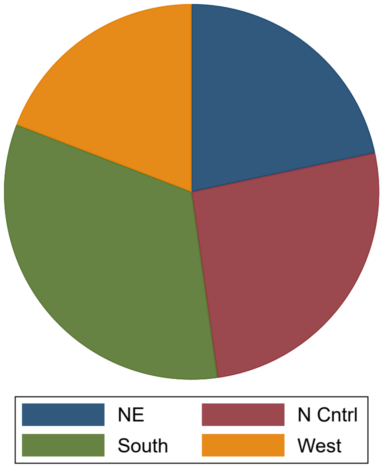
```
graph dot popk, over(division)
```

Here the previous plot is shown as a dot plot by using `graph dot`. Dot plots are often used to show one or more summary statistics for one or more continuous variables, broken down by one or more categorical variables. See Dot (section 6) for more details. *Uses allstates.dta*



```
graph pie popk, over(region)
```

The `graph pie` command creates a pie chart. See Pie (section 7) for more details. *Uses allstates.dta*



1.4 Schemes

A scheme specifies the overall look of a graph. Stata comes with several different schemes that you can choose from. This section illustrates the look of a basic bar graph when shown using some commonly used schemes that come with Stata (see section 1.4.1). Next, I show this same bar graph when using selected schemes developed by members of the Stata community (see section 1.4.2.) The following section illustrates graphs created using schemes I developed for this book (see section 1.4.3). This section concludes with some additional details about specifying schemes (see section 1.4.4).

This section gives you just a brief introduction to schemes, trying to give you a flavor for how they work. You can find more details later in this book in Standard options : Schemes (section 9.2). Also, to download the datasets and schemes used in this book, see Introduction : Online supplements (section 1.1).

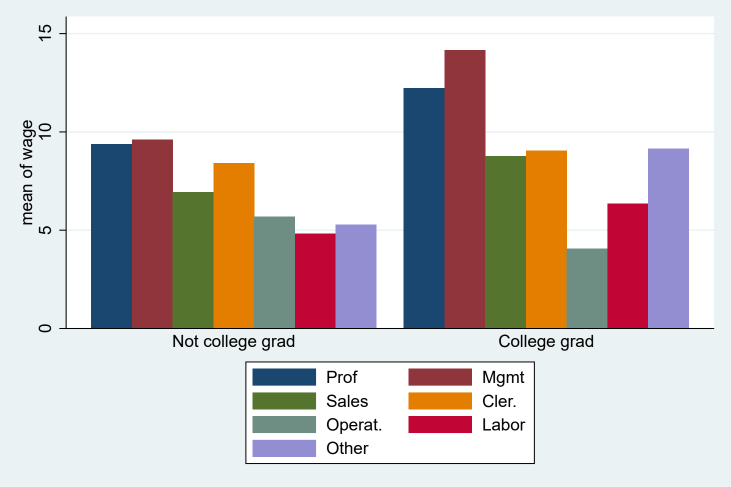
Graph jargon. I will often comment on whether shading is present in the *plot region* or the *graph region*. In the bar graph below, the *plot region* is the white part of the graph where the bars are plotted. The *graph region* is the light-blue area where the axis labels, axis titles, and graph titles are displayed. In the example below, the plot region is white and includes horizontal grid lines. The graph region is shown in light blue, and the legend is displayed with a white background. These kinds of details are specified by the `scheme()`, and we will examine those details as we compare graphs created using different schemes.

1.4.1 Schemes included with Stata

The following examples illustrate commonly used schemes that are included with Stata, including `s1color`, `s1mono`, `s2color`, `s2mono`, `s1rcolor`, and `economist`.

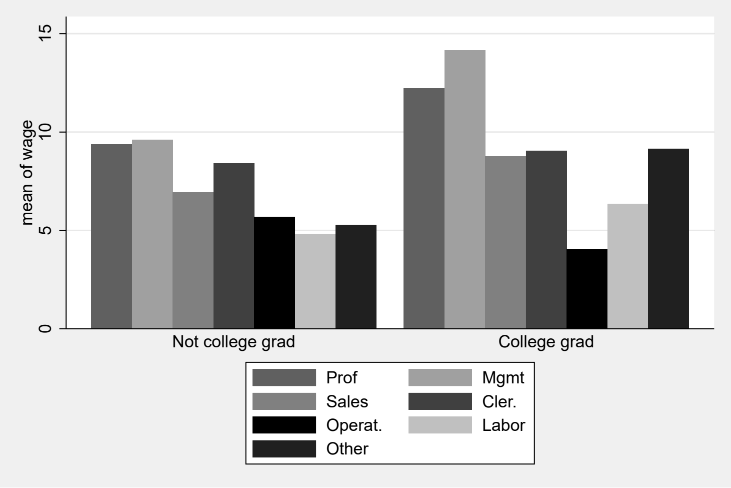
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s2color)
```

This bar chart shows an example of a graph created using the `s2color` scheme. The plot region has a white background, but the surrounding area (the graph region) is light blue. The bars are displayed using colors specified by the `s2color` scheme. *Uses nlsw.dta*



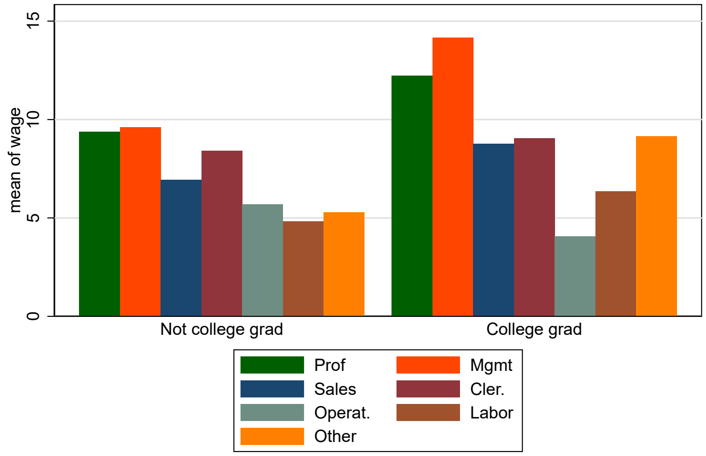
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s2mono)
```

This bar chart is identical to the prior example, except that it uses the `s2mono` scheme. This is the monochrome equivalent of the `s2color` scheme. This scheme is in black and white, and the plot region has a white background, but the surrounding area (the graph region) is light gray. The bars are displayed using different shades of gray. *Uses nlsw.dta*



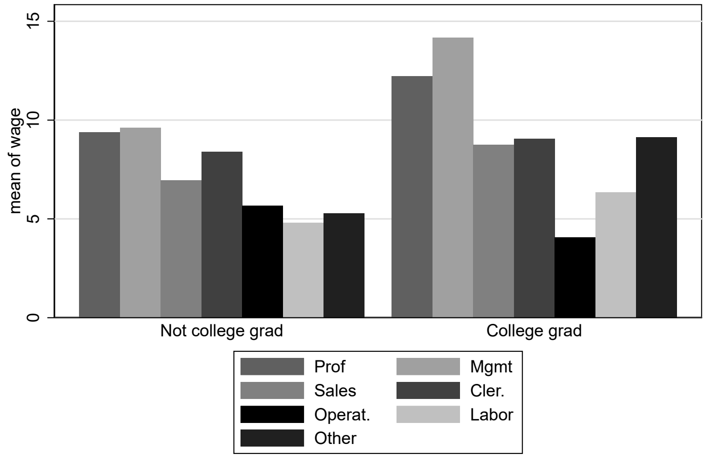
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s1color)
```

This `graph bar` uses the `scheme(s1color)` option. The `s1` family of schemes, in contrast to the `s2` family, uses a white background for both the plot region and graph region (the area surrounding the plot region). Also, the bar colors used by `s1color` are different from the bar colors used by the `s2color` scheme. *Uses nlsw.dta*



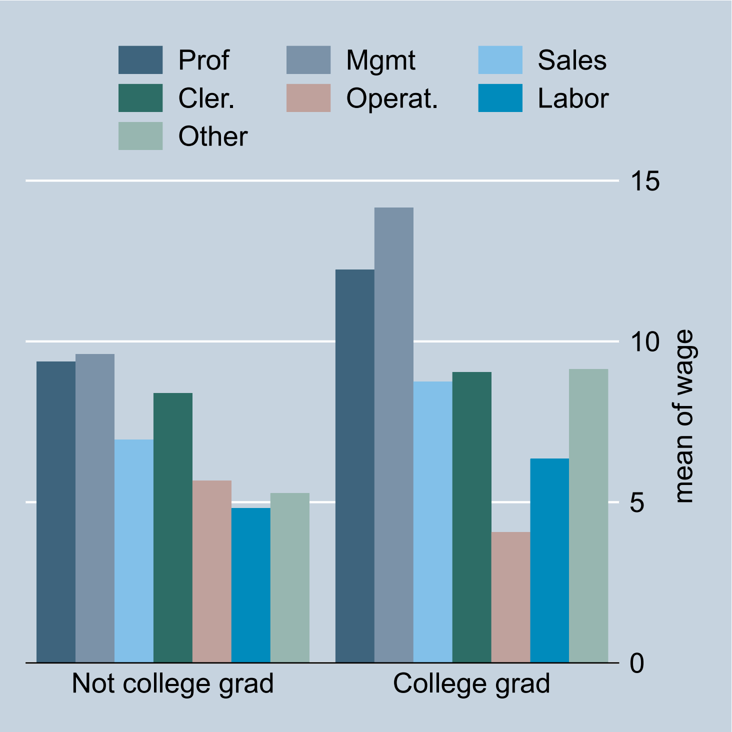
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s1mono)
```

This bar chart is displayed using the `s1mono` scheme. This is the monochrome equivalent of the `s1color` scheme. These bars are shown in black and white against a white background. Both the plot region and the graph region are white. *Uses nlsw.dta*



```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(economist)
```

This `graph bar` command produces an example of a graph created using the `economist` scheme. This scheme produces graphs that look similar to those that appear in *The Economist* magazine. This illustrates that a scheme can have a considerable impact on the look of a graph. In this case, the legend is displayed at the top, the y axis is labeled at the right, and the entire graph has a grayish background. Also, the bar colors are notably different. *Uses nlsw.dta*

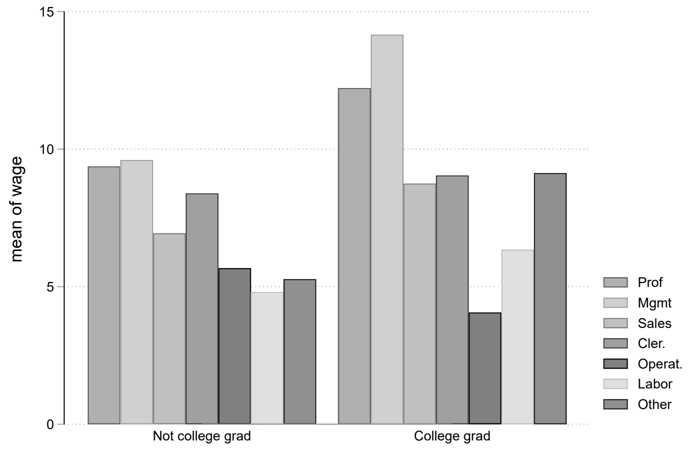


1.4.2 Community-contributed schemes

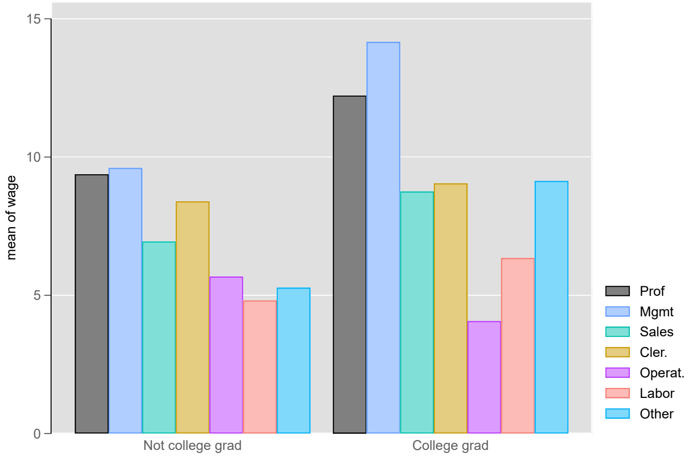
In this next set of examples, we will use the same `graph bar` command, using selected schemes developed by members of the Stata community from around the world. Each example illustrates the look provided by the specified scheme when creating a bar graph. Section 9.2.2 provides more information about these schemes, describes how to download them, and then shows additional example graphs created using these schemes.

```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(plotplain)
```

This is an example of a bar chart that uses the `plotplain` scheme. True to its name, this scheme creates graphs that are plain and clear. The bars are displayed using different shades of gray, and the legend is compactly displayed at the right. Also, both the plot region and the surrounding graph region (where the titles and axis labels are displayed) are both white. See section 9.2.2 for information about this scheme, including how to download it. *Uses nlsw.dta*



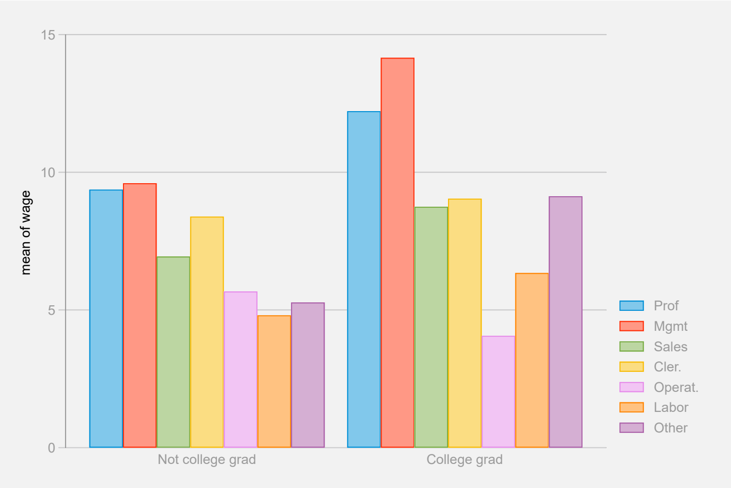
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(plottig)
```



This bar chart specifies the `scheme(plottig)` option, creating a chart that uses the `plottig` scheme. The look of this scheme is modeled after `ggplot2` (in R). We can see the bars are shown using different colors, with a compact legend to the right. The plot region is shown in gray and the surrounding graph region (where the titles and axis labels are displayed) is white. See section 9.2.2 for information about this scheme, including how to download it. *Uses nlsw.dta*

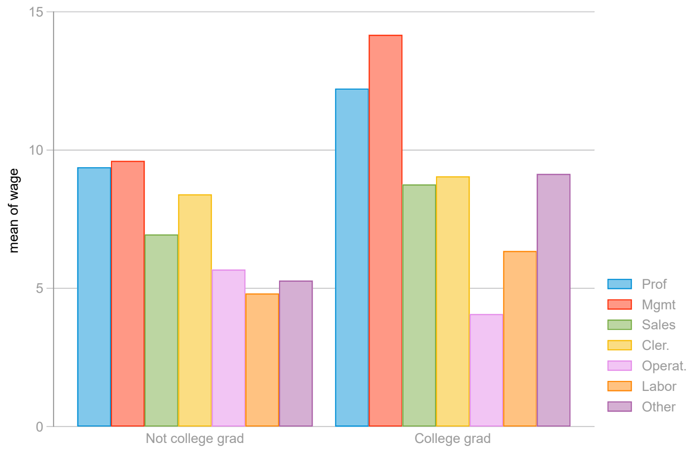
```
graph bar wage, over(occ7) over(collgrad) asyvars scheme(538)
```

This bar chart illustrates the look of a graph using the `538` scheme, modeled after the design of figures on the website `https://fivethirtyeight.com`. The bars are displayed using different colors, with a compact legend at the right. The plot region and surrounding graph region are gray. See section 9.2.2 for information about this scheme, including how to download it. *Uses nlsw.dta*



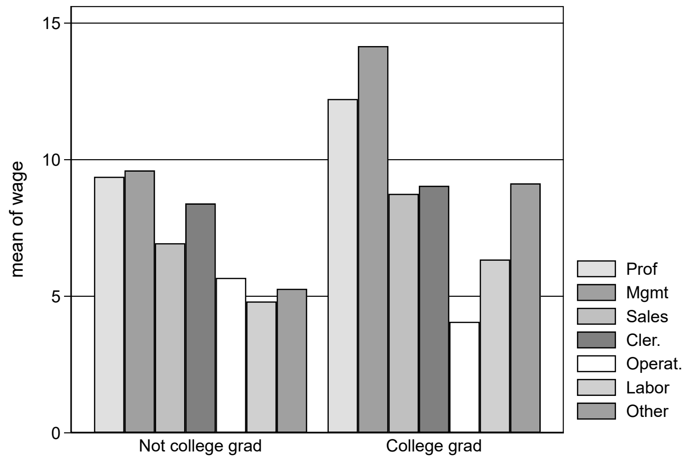
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(538w)
```

This graph was created using the `scheme(538w)` option. As you can see, this graph looks identical to the prior graph, except that the entire background color is white. See section 9.2.2 for information about this scheme, including how to download it. *Uses nlsw.dta*



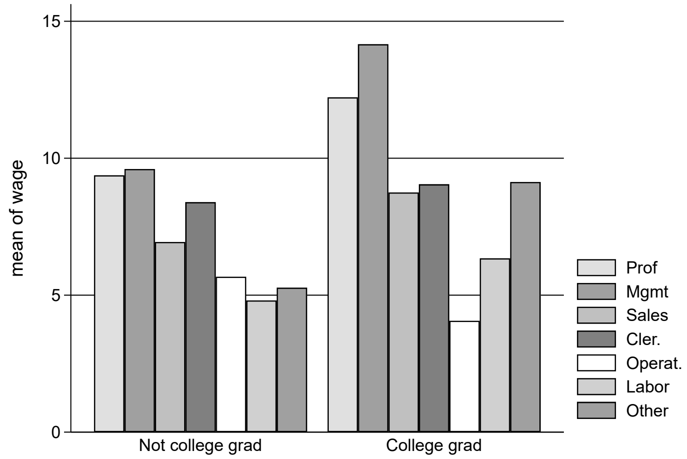
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(lean1)
```

This bar chart illustrates the look of the `lean1` scheme. Note how the bars are displayed using different shades of gray. The legend is displayed at the right. The plot region is framed, and the plot and graph regions are white. See section 9.2.2 for information about this scheme, including how to download it. *Uses nlsw.dta*



```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(lean2)
```

This graph shows the kind of bar chart produced using the `lean2` scheme. The look of this graph is very similar to the prior graph. The main difference is that the plot region in this example is not framed. See section 9.2.2 for information about this scheme, including how to download it. *Uses nlsw.dta*

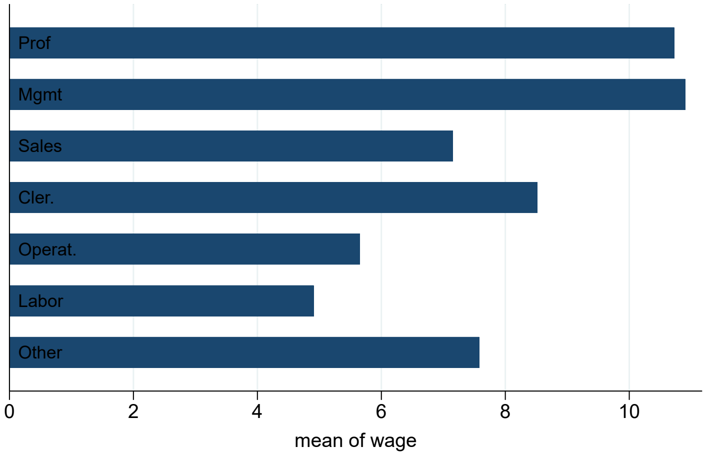


1.4.3 Schemes included with this book

This section illustrates schemes included in this book. The prior schemes I illustrated have a distinct style and provide advantages in a variety of contexts. The schemes I developed are like one of those kitchen devices that do one thing (an apple corer or ice cream scooper). So the following examples illustrating the schemes from this book will show off the one thing that scheme does well, and it will point you to an example inside the body of the book so you can see that example in context. The index shows all instances where such schemes are used in this book; for example, the index entry “schemes, `vg_lgndc`” shows each example in this book where this scheme was used. See Introduction : Online supplements (section 1.1) for information about downloading the data and schemes from this book, and see Standard options : Schemes (section 9.2) for more details about schemes.

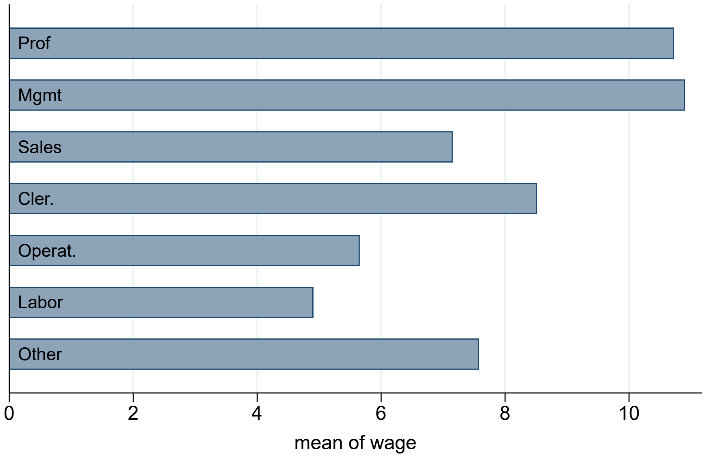
```
graph hbar wage, over(occ7, label(nolabels)) blabel(group,
position(base))
```

Consider this horizontal bar chart. We include the `blabel(group, position(base))` option to display the group names for each bar at the base of each chart. However, because the bars are so dark, it is hard to read the labels for each bar. *Uses nlsw.dta*



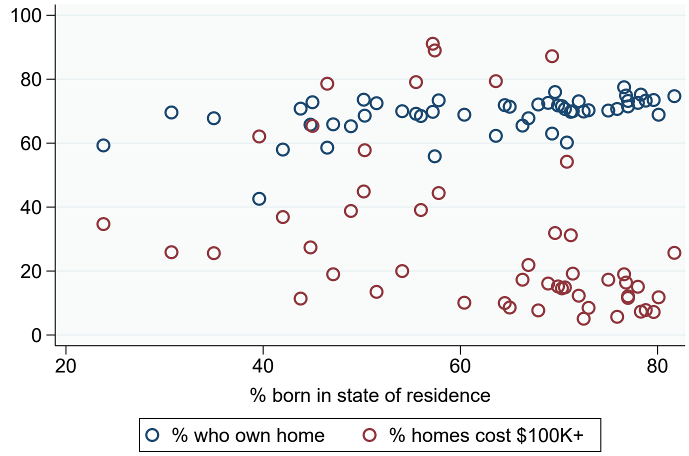
```
graph hbar wage, over(occ7, label(nolabels)) blabel(group,
position(base)) scheme(vg_palec)
```

By adding the `scheme(vg_palec)` scheme to the graph command from above, colors of the bars are paler (have less intensity). With this scheme, the colors of the bars are pale enough to include text labels inside bars. This example is illustrated in the book, in



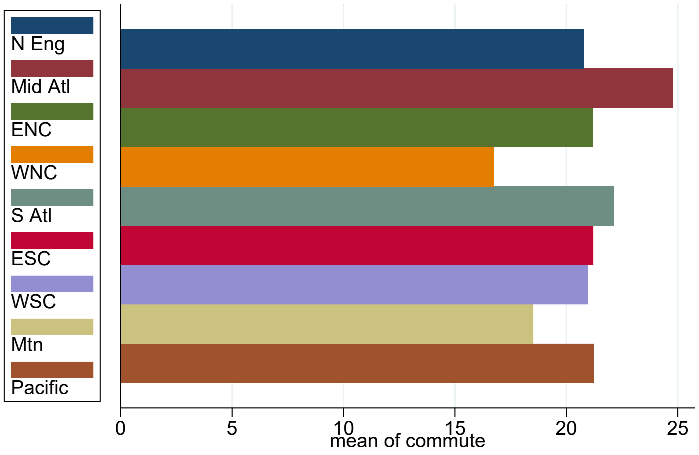
context, in section 4.6 . The monochrome equivalent of this scheme is `vg_palem` (not illustrated). *Uses nlsw.dta*

```
twoway scatter ownhome propval100 borninstate,
scheme(vg_hollowc)
```



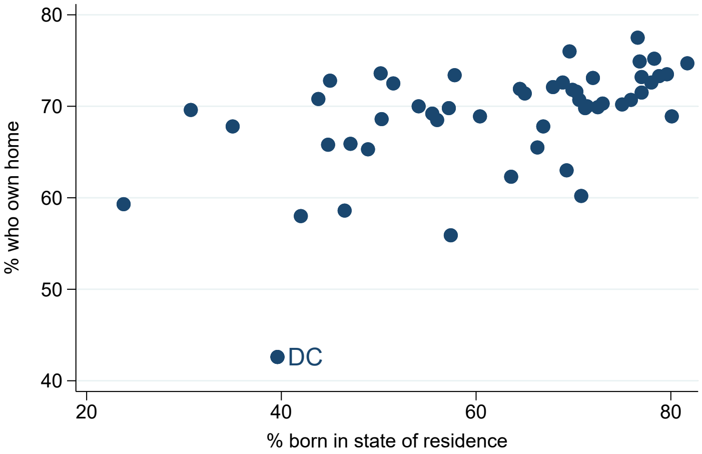
When several markers are clustered together, this can sometimes create overlap among solidly filled markers. By using the `scheme(vg_hollowc)` scheme, you can make the fill color for markers invisible. As you can see, using an invisible fill can aid in seeing markers that are clumped together or nearly overlapping. You can see this example, in context, in section 8.1. There is a monochrome equivalent of this scheme named `vg_hollowm` (not illustrated). *Uses allstates.dta*

```
graph hbar commute, over(division) asyvar scheme(vg_lgndc)
```



When creating this horizontal bar chart, we use the `vg_lgndc` scheme because it placed the legend to the left of the graph using one column. You can see this scheme used in context for creating bar charts (see section 4.6), for creating box plots (see section 5.5), and for creating pie charts (see section 7.2). For more examples, see the index entry for “scheme, `vg_lgndc`". The monochrome version, `vg_lgndm`, is not illustrated to save space. *Uses allstates.dta*

```
twoway (scatter ownhome borninstate if stateab=="DC",
mlabel(stateab)) (scatter ownhome borninstate), legend(off)
scheme(vg_samec)
```



This graph illustrates use of the `vg_samec` scheme. This scheme, based on the `s2color` scheme, makes all markers, lines, bars, etc., the same color, shape, and pattern. The second `scatter` command labels Washington DC, which normally would be shown in a different color; with this scheme, the marker is the same. For examples in context, see section 2.1 . The monochrome equivalent, called `vg_samem`, is not illustrated to save space. *Uses allstates.dta*

1.4.4 Setting schemes

As these examples have shown, we can change the scheme of a graph by supplying the `scheme()` option on a graph command. If you want to use the same scheme over and over (for example, `economist`), you can use the `set scheme` command below to set the default scheme to the `economist` scheme.

```
. set scheme economist
```

This would persist until you quit Stata. Or you could type

```
. set scheme economist, permanently
```

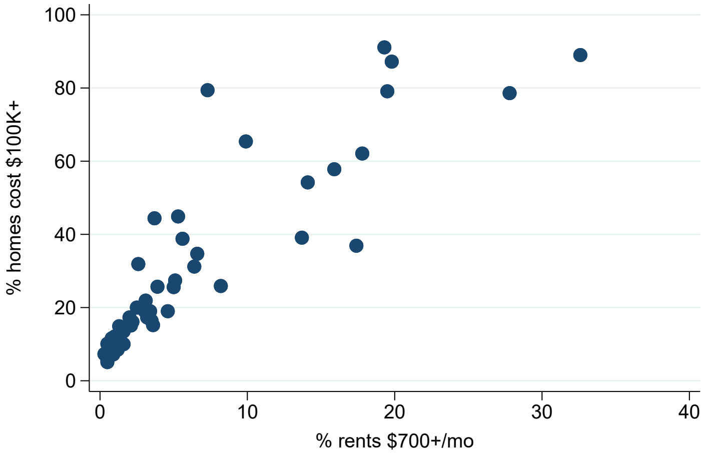
and the `economist` scheme would be your default scheme, even after you quit and restart Stata. The `set scheme` command is useful if you want to create a series of commands with the same scheme without having to repeatedly use the `scheme()` option. See Standard options : Schemes (section 9.2) for additional details about schemes.

1.5 Options

Learning to create effective Stata graphs is ultimately about using options to customize the look of a graph until you are pleased with it. This section illustrates the general rules and syntax for Stata graph commands, starting with their basic structure and followed by illustrations showing how options work in the same way across different kinds of commands. Stata graph options work much like other options in Stata; however, there are more features that extend their power and functionality. Although these examples will use the `twoway scatter` command for illustration, most of the principles illustrated extend to all kinds of Stata graph commands.

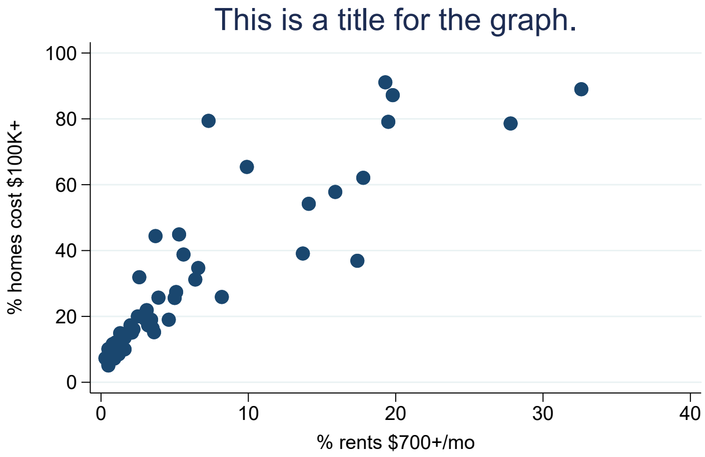
```
twoway scatter propval100 rent700
```

Consider this basic scatterplot. To add a title to this graph, we can use the `title()` option as illustrated in the next example. *Uses allstates.dta*



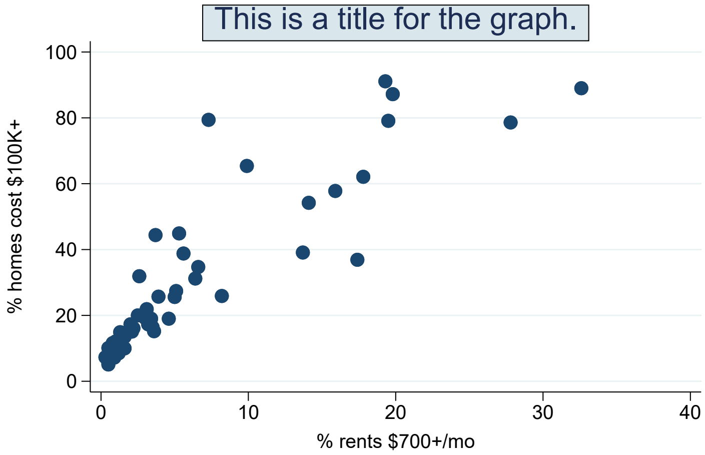
```
twoway scatter propval100 rent700, title("This is a title for
the graph.")
```

Just as with any Stata command, the `title()` option comes after a comma, and here it contains a quoted string that becomes the title of the graph. *Uses allstates.dta*



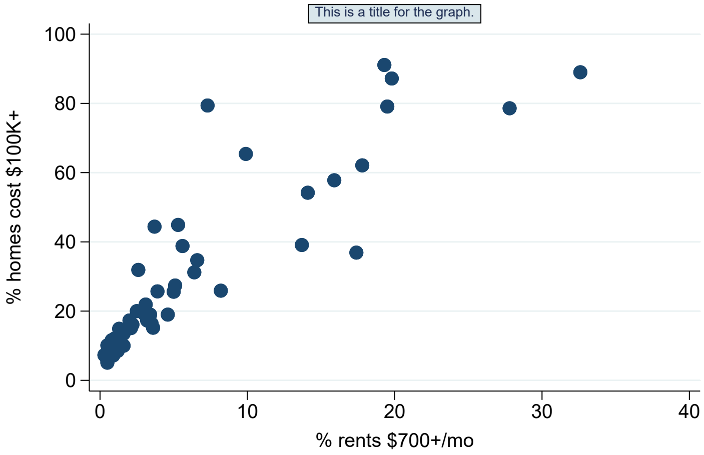
```
twoway scatter propval100 rent700, title("This is a title for
the graph.", box)
```

In this example, we add `box` as an option within `title()` to place a box around the title. If the default for the current scheme had included a box, then you could have used the `nobox` option to suppress it. *Uses allstates.dta*

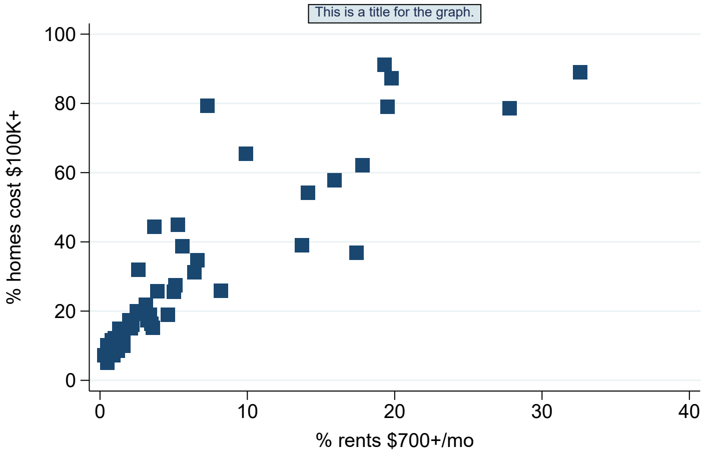


```
twoway scatter propval100 rent700, title("This is a title for
the graph.", box size(small))
```

Let's take the last graph and modify the title to make it small. We add `size(small)` to the `title()` option to change the title's size, as well as the box size, to small. *Uses allstates.dta*



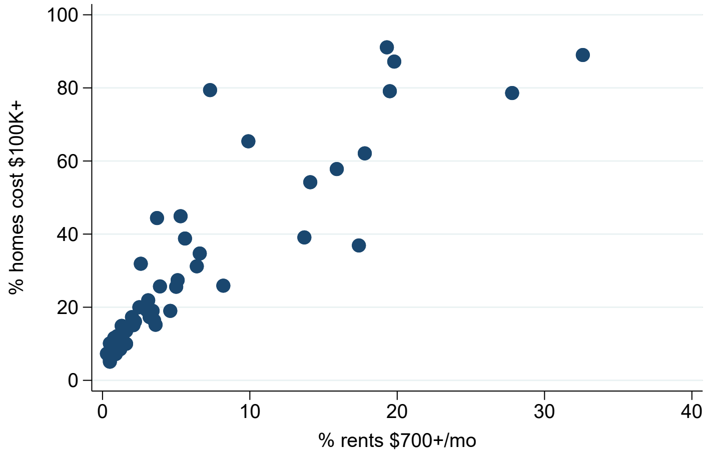
```
twoway scatter propval100 rent700, title("This is a title for
the graph.", box size(small)) msymbol(S)
```



Say that we want the symbols to be displayed as squares. We add another option, `msymbol(S)`, to indicate that we want the marker symbol to be displayed as a square (`S` for square). Adding one option at a time is a common way to build a Stata graph. In the next graph, we will change gears and start building a new graph to show other aspects of options. *Uses allstates.dta*

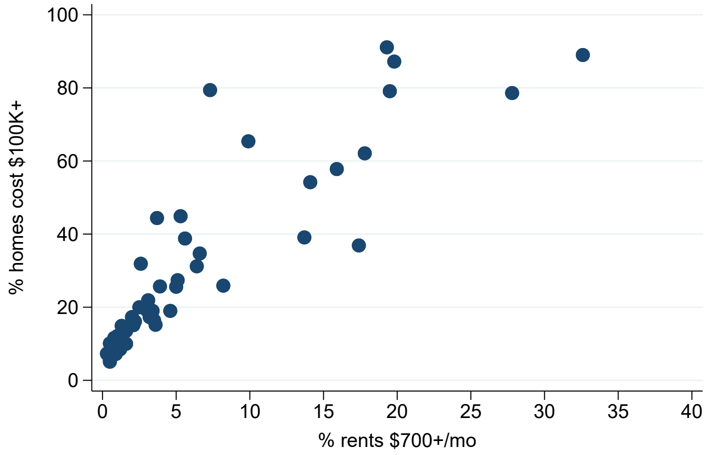
```
twoway scatter propval100 rent700
```

Let's return to this simple scatterplot. Say that we want the labels for the x axis to change from 0 10 20 30 40 to 0 5 10 15 20 25 30 35 40. *Uses allstates.dta*



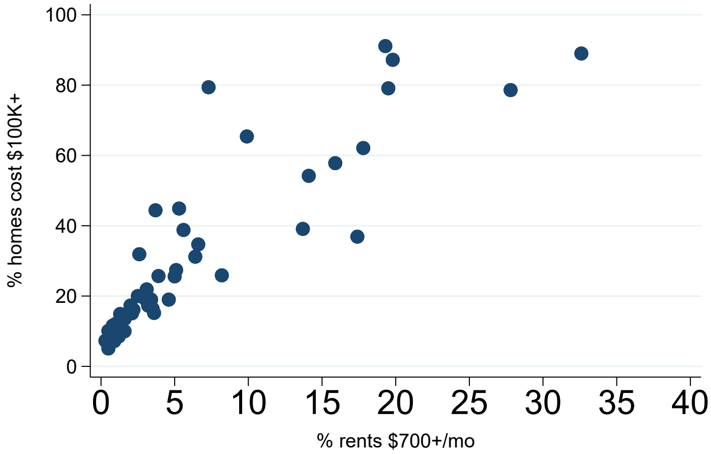
```
twoway scatter propval100 rent700, xlabel(0(5)40)
```

Here we add the `xlabel()` option to label the x axis from 0 to 40, incrementing by 5. But say that we want the labels to be displayed larger. *Uses allstates.dta*



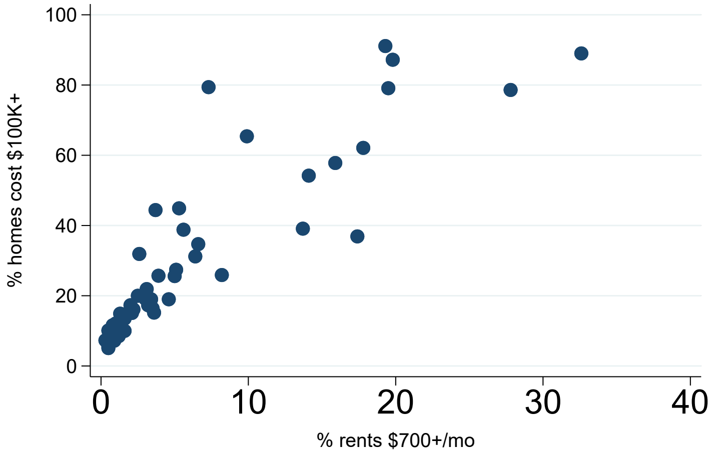
```
twoway scatter propval100 rent700, xlabel(0(5)40,
labsize(huge))
```

Here we add the `labsize()` (label size) option to increase the size of the labels for the x axis. Now say that we were happy with the original numbering (0 10 20 30 40) but wanted the labels to be huge. *Uses allstates.dta*



```
twoway scatter propval100 rent700, xlabel(, labsize(huge))
```

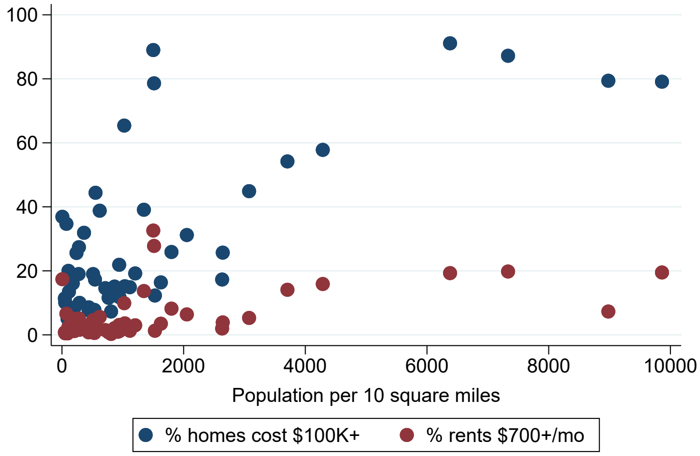
The `xlabel()` option we use here indicates that we are content with the numbers chosen for the label of the x axis because we have nothing before the comma. After the comma, we add the `labsize()` option to increase the size of the labels for the x axis. *Uses allstates.dta*



Now let's consider some examples using the `legend()` option to show that some options do not require or permit the use of commas within them. Also, the following examples show where we might properly specify an option repeatedly.

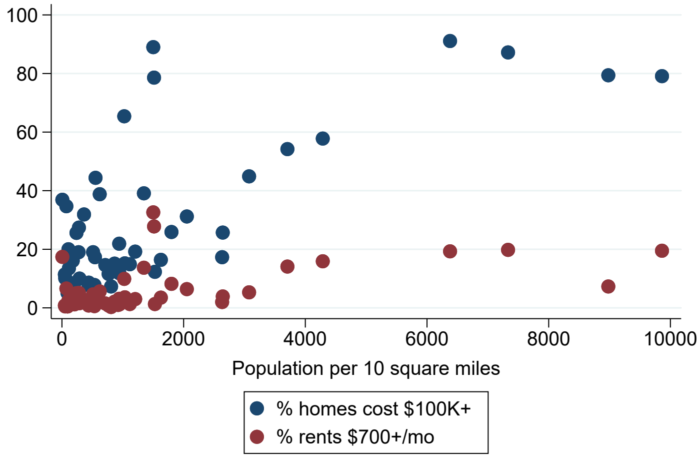
```
twoway scatter propval100 rent700 popden
```

Here we show two y variables, `propval100` and `rent700`, graphed against population density, `popden`. Stata has created a legend, helping us see which symbols correspond to which variables. We can use the `legend()` option to customize the legend. *Uses allstates.dta*



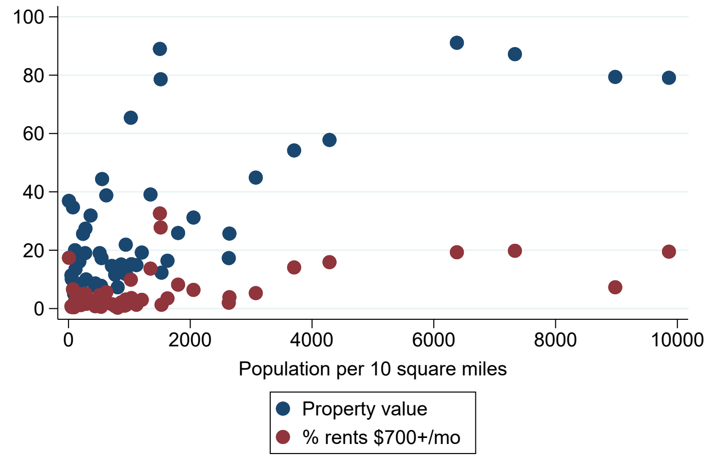
```
twoway scatter propval100 rent700 popden, legend(cols(1))
```

By using the `legend(cols(1))` option, we make the legend display in one column. We did not use a comma because, with the `legend()` option, there is no natural default argument. If we had included a comma within the `legend()` option, Stata would have reported this as an error. *Uses allstates.dta*

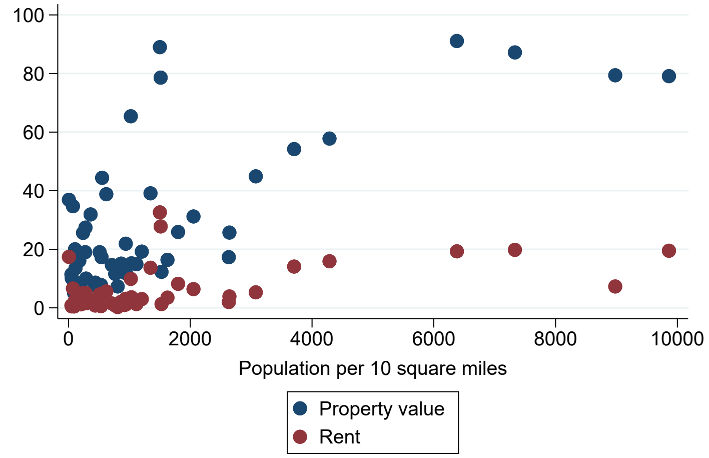


```
twoway scatter propval100 rent700 popden, legend(cols(1)
label(1 "Property value"))
```

This example adds another option, `label()`, within the `legend()` option to change the label for the first variable. *Uses allstates.dta*



```
twoway scatter propval100 rent700 popden, legend(cols(1)
label(1 "Property value") label(2 "Rent"))
```

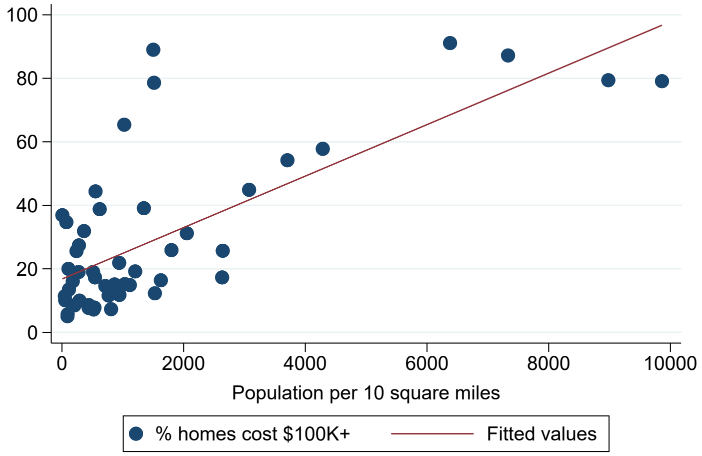


Here we add another `label()` option within the `legend()` option; this option changes the label for the second variable. We can use the `label()` option repeatedly to change the labels for the different variables. *Uses allstates.dta*

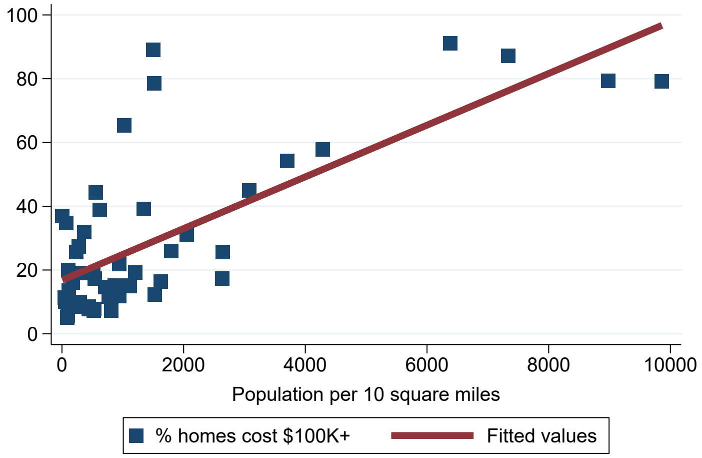
Finally, let's consider an example that shows how to use the `twoway` command to overlay two plots. The following examples show how each graph can have its own options and how options can apply to the overall graph.

```
twoway (scatter propval100 popden) (lfit propval100 popden)
```

This graph shows a scatterplot predicting property value from population density and shows a linear fit between these two variables. Say that we wanted to change the symbol displayed in the scatterplot and the thickness of the line for the linear fit. *Uses allstates.dta*



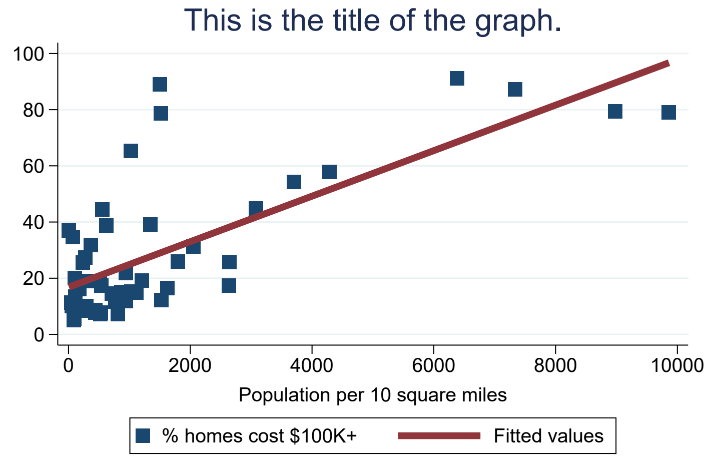
```
twoway (scatter propval100 popden, msymbol(S)) (lfit
propval100 popden, lwidth(vthick))
```



We add the `msymbol()` option to the `scatter` command to change the symbol to a square, and we add the `lwidth()` (line width) option to the `lfit` command to make the line very thick. When we overlay two plots, each plot can have its own options that operate on its respective parts of the graph. However, some parts of the graph are shared, for example, the title. *Uses allstates.dta*

```
twoway (scatter propval100 popden, msymbol(S)) (lfit
propval100 popden, lwidth(vthick)), title("This is the title
of the graph.")
```

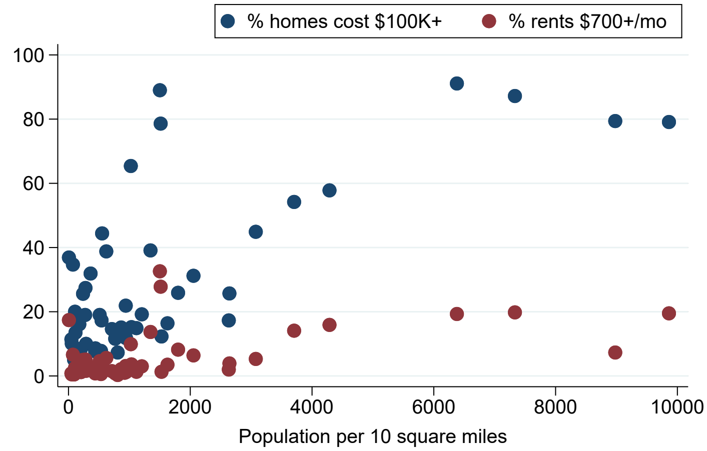
We add the `title()` option to the end of the command, placed after a comma. That final comma signals that options concerning the overall graph are to follow, here, the `title()` option. *Uses allstates.dta*



One of the beauties of Stata graph commands is the way that different graph commands share common options. If you want to customize the display of a legend, you do it by using the same options, whether you are using a bar graph, a box plot, a scatterplot, or any other kind of Stata graph. Once you learn how to control the legend with one type of graph, you have learned how to control the legend for all types of graphs. This is illustrated with a couple of examples.

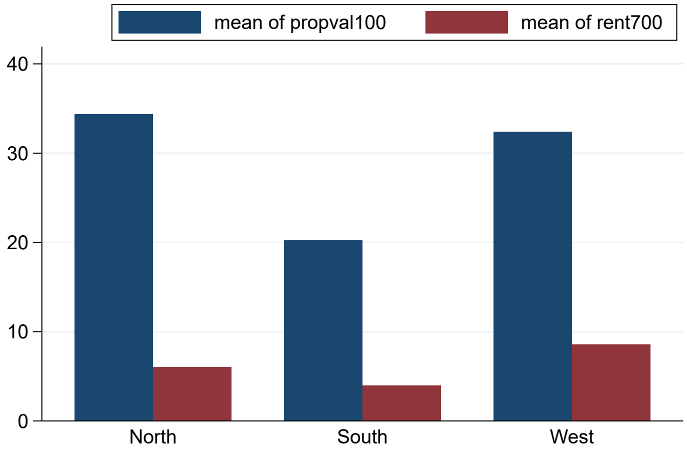
```
twoway scatter propval100 rent700 popden, legend(position(1))
```

Consider this scatterplot. We add the `legend()` option to make the legend display in the one o'clock position on the graph, putting the legend in the top right corner. *Uses allstates.dta*



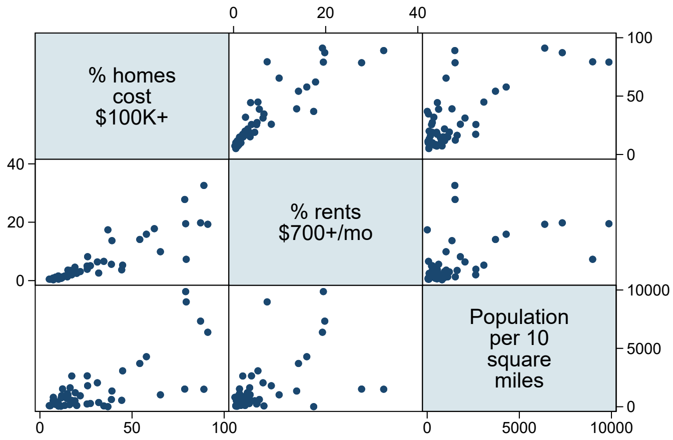
```
graph bar propval100 rent700, over(nsw) legend(position(1))
```

Here we use the `graph bar` command, which is a completely different command from the previous one. Even though the graphs are different, the `legend()` option we supply is the same and has the same effect. Many (but not all) options function in this way, sharing a common syntax and having common effects. *Uses allstates.dta*



```
graph matrix propval100 rent700 popden, legend(position(1))
```

Compare this example with the previous two. The `graph matrix` command does not support the `legend()` option because this graph does not need or produce a legend. In the **Matrix** (section 3) chapter, for example, there are no references to a legend, an indication that this is not a relevant option for this kind of graph. Even though we included this irrelevant option, Stata ignored it and produced an appropriate graph anyway. *Uses allstates.dta*



Because the legend works the same way with different types of Stata graph commands, only one place discusses the legend in detail: **Options : Legend** (section 8.9). However, it is useful to see examples of the legend for each type of graph that uses them. Each chapter, therefore, includes a brief section describing the legend for each type of graph discussed in that chapter. Likewise, **Options** (section 8) describes most options in detail, with a brief section in every chapter discussing how each option works for that specific type of graph. As shown for the legend, some options are not appropriate for some types of graphs, so those options are not discussed with the commands that do not support them.

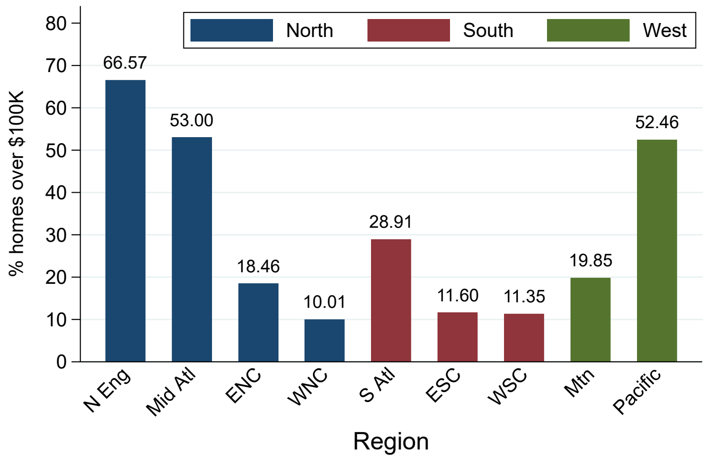
Although you can use an option like `legend()` with many, but not all, kinds of Stata graph commands, you can use other kinds of options with almost every kind of Stata graph. These are called *standard options*. To help you differentiate these kinds of options, they are discussed in their own chapter, Standard options (section 9). Because these options can be used with most types of graph commands, they are generally not discussed in the chapters about the different types of graphs, except when their usage interacts with the options illustrated. For example, `subtitle()` is a standard option, but its behavior takes on a special meaning when used with the `legend()` option, so the `subtitle()` option is discussed in the context of the legend. Consistent with what was previously shown, the syntax of standard options follows the same kinds of rules that have been illustrated, and their usage and behavior are uniform across the many types of Stata graph commands.

1.6 Building graphs

I have three agendas in writing this section. First, I wish to show the process of building complex graphs a little bit at a time. At the same time, I illustrate how to use the resources of this book to get the bits of information needed to build these graphs. Finally, I hope to show that, even though a complete Stata graph command might look complicated and overwhelming, the process of building the graph slowly is actually straightforward and logical.

Let's first build a bar chart that looks at property values broken down by region of the country. Then we will modify the legend and bar characteristics, add titles, and so forth.

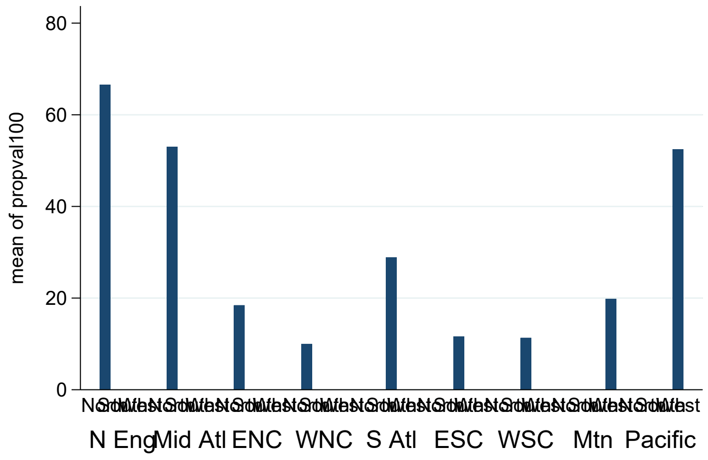
```
graph display
```



Say that we want to create this graph. For now, the syntax is concealed, just showing the `graph display` command to show the previously drawn graph. It might be overwhelming at first to determine all the options needed to make this graph. To ease our task, we will build it a bit at a time, refining the graph and fixing any problems we find. *Uses allstates.dta*

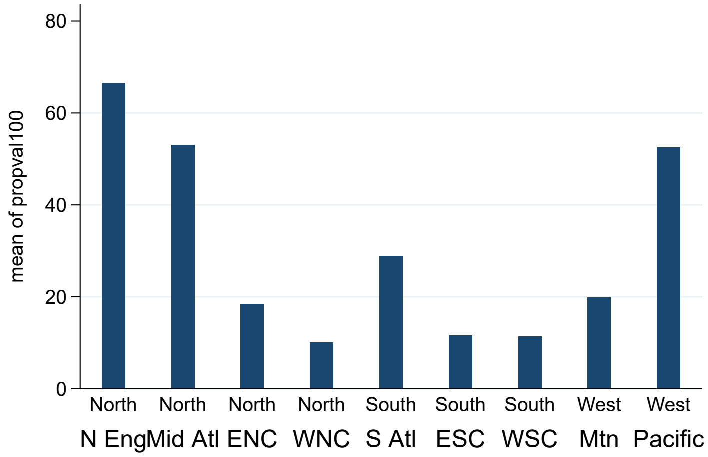
```
graph bar propval100, over(nsw) over(division)
```

We begin by seeing that this is a bar chart and look at Bar : Y-variables (section 4.1) and Bar : Over (section 4.2). We take our first step toward making this graph by making a bar chart showing `propval100` and adding `over(nsw)` and `over(division)` to break down the means by `nsw` and `division`. *Uses allstates.dta*



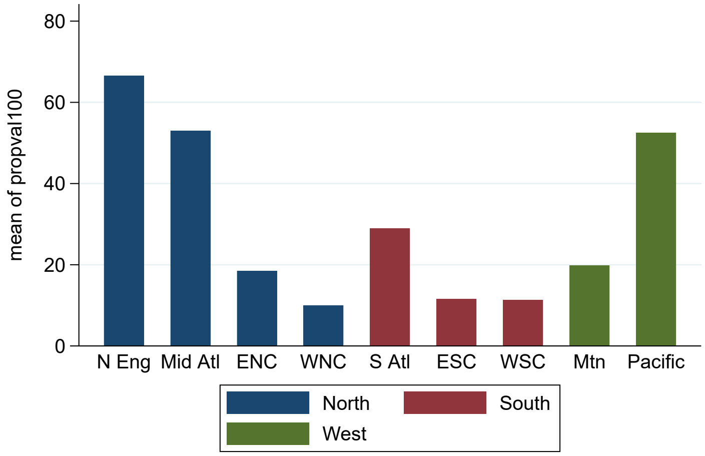
```
graph bar propval100, over(nsw) over(division) nofill
```

The previous graph is not quite what we want because we see every `division` shown with every `nsw`, but for example, the Pacific region only appears in the West. In **Bar : Over** (section 4.2), we see that we can add the `nofill` option to show only the combinations of `nsw` and `division` that exist in the data file. Next we will look at the colors of the bars. *Uses allstates.dta*



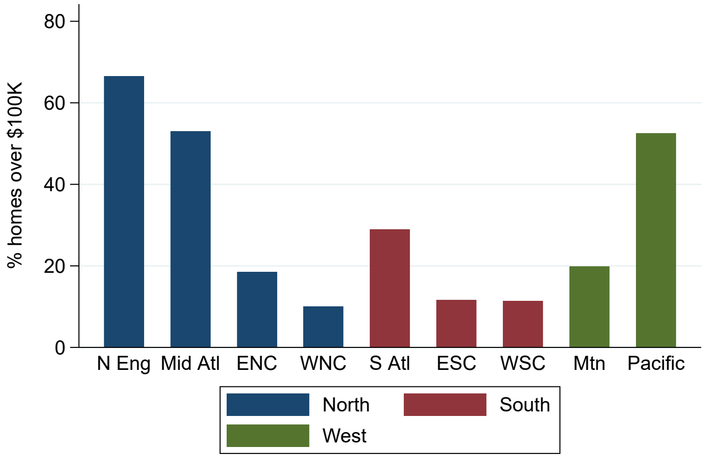
```
graph bar propval100, over(nsw) over(division) nofill asyvars
```

The last graph is getting closer, but we want the bars for North, South, and West displayed in different colors and labeled with a legend. In **Bar : Y-variables** (section 4.1), we see that the `asyvars` option will accomplish this. Next we will change the title for the y axis. *Uses allstates.dta*



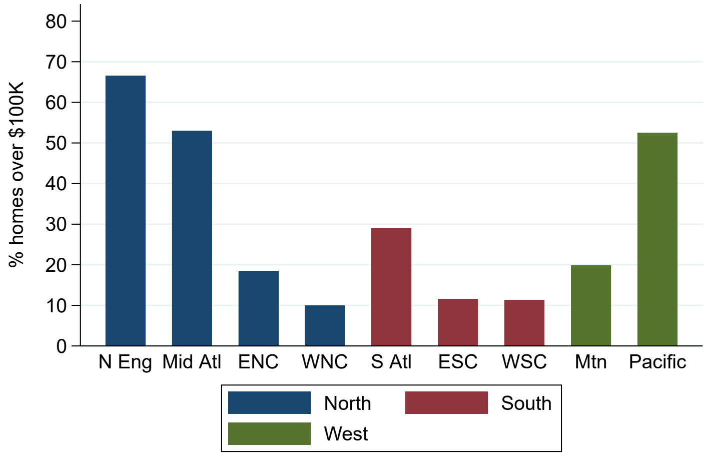
```
graph bar propval100, over(nsw) over(division) nofill asyvars
ytitle("% homes over $100K")
```

Now we want to put a title on the y axis. In **Bar : Y-axis** (section 4.7), we see examples illustrating the use of `ytitle()` for putting a title on the y axis. Here we put a title on the y axis, but now we want to change the labels for the y axis to go from 0 to 80, incrementing by 10. *Uses allstates.dta*

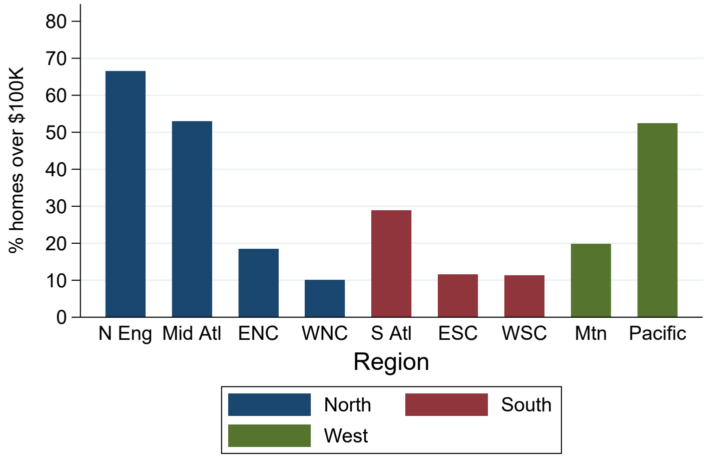


```
graph bar propval100, over(nsw) over(division) nofill asyvars
ytitle("% homes over $100K") ylabel(0(10)80)
```

The **Bar : Y-axis** (section 4.7) section also tells us about the `ylabel()` option. Now that we have the y axis labeled as we want, let's next look at the title for the x axis. *Uses allstates.dta*



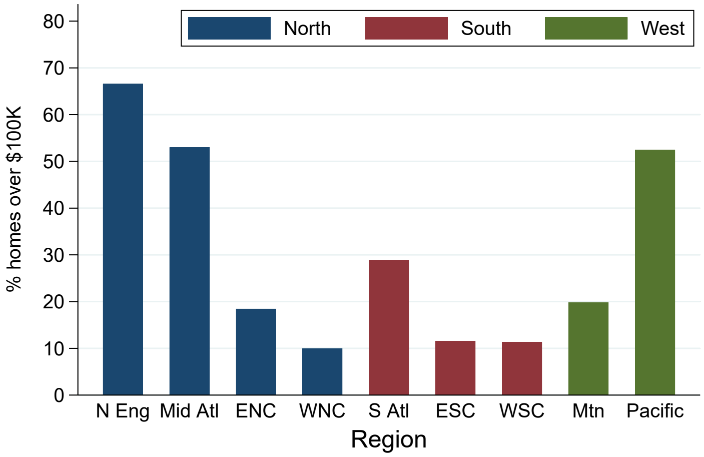
```
graph bar propval100, over(nsw) over(division) nofill asyvars
ytitle("% homes over $100K") ylabel(0(10)80) b1title(Region)
```



After having used the `ytitle()` option to label the y axis, we might be tempted to use the `xtitle()` option to label the x axis, but this axis is a categorical variable. In Bar : Cat axis (section 4.5), we see that this axis is treated differently because of that. To put a title below the graph, we use the `b1title()` option. Now let's turn our attention to formatting the legend. *Uses allstates.dta*

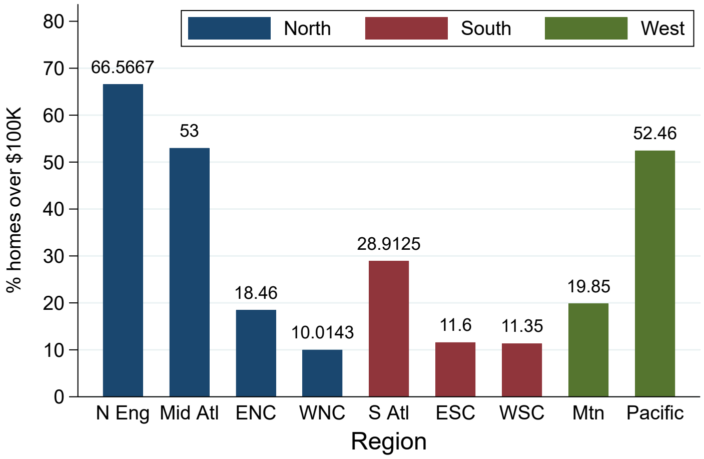
```
graph bar propval100, over(nsw) over(division) nofill asyvars
ytitle("% homes over $100K") ylabel(0(10)80) b1title(Region)
legend(rows(1) position(1) ring(0))
```

Here we want to use the `legend()` option to make the legend have one row in the top right corner within the plot area. In Bar : Legend (section 4.6), we see that the `rows(1)` option makes the legend appear in one row and that the `position(1)` option puts the legend in the one o'clock position. The `ring(0)` option puts the legend inside the plot region. Next let's label the bars. *Uses allstates.dta*



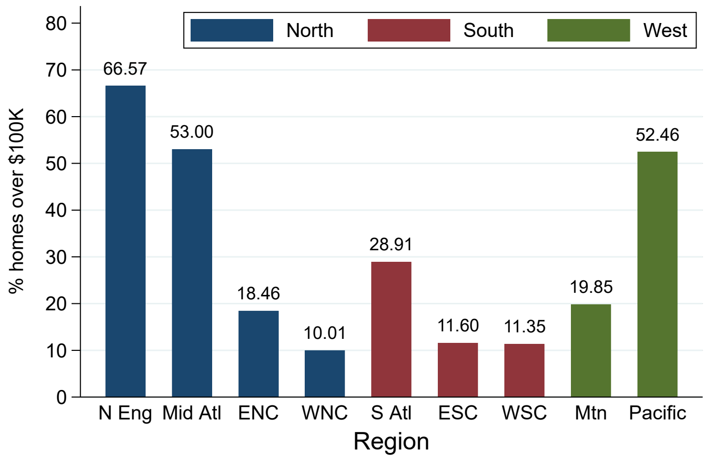
```
graph bar propval100, over(nsw) over(division) nofill asyvars
ytitle("% homes over $100K") ylabel(0(10)80) b1title(Region)
legend(rows(1) position(1) ring(0)) blabel(bar)
```

We want each bar labeled with the height of the bar. Bar : Legend (section 4.6) shows how we can do this by using the `blabel()` (bar label) option to label the bars in lieu of a legend. `blabel(bar)` labels the bars with their height. *Uses allstates.dta*



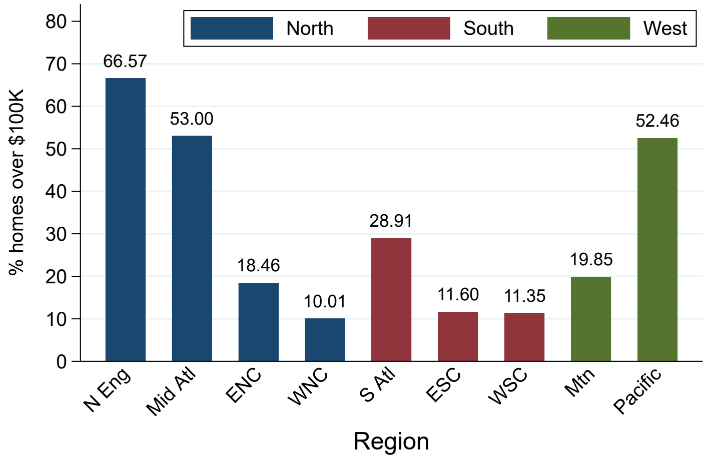
```
graph bar propval100, over(nsw) over(division) nofill asyvars
ytitle("% homes over $100K") ylabel(0(10)80) b1title(Region)
legend(rows(1) position(1) ring(0)) blabel(bar,
format(%4.2f))
```

We want the label for each bar to end in two decimal places, and we see in Bar : Legend (section 4.6) that we can use the `format()` option to format these numbers as we want. *Uses allstates.dta*



```
graph bar propval100, over(nsw) over(division,
label(angle(45))) nofill ytitle("% homes over $100K")
ylabel(0(10)80) b1title(Region) legend(rows(1) position(1)
ring(0)) blabel(bar, format(%4.2f)) asyvars
```

Finally, in Bar : Cat axis (section 4.5), we see that we can add the `label(angle(45))` option to the `over()` option to specify that labels for that variable be shown at a 45-degree angle so they do not overlap each other. *Uses allstates.dta*



I hope this section has shown that it is not that difficult to create complex graphs by building them one step at a time. You can use the resources in this book to seek out each piece of information you need and then put those pieces together the way you want to create your own graphs. For more information about how to integrate options to create complex Stata graphs, see Appendix : More examples (section 11.6).

1.7 Point-and-click interface

Many people have an aversion to the use of point-and-click methods for creating statistical results or statistical figures. A key part of this aversion is that such methods are frequently not repeatable, violating a key scientific principle of repeatability. However, the Stata point-and-click interface produces the commands that can be used to replicate each result/graph it creates. Because of this, the Stata point-and-click interface offers the advantages of an interactive point-and-click interface combined with repeatability. In this section, I would like to highlight some of the advantages of the point-and-click interface for the creation and customization of graphs.

Teaching about a dynamic point-and-click interface via a static medium like a book is very difficult. Instead, a point-and-click interface is taught more effectively via online videos, like the ones that you can find in the Stata Video Tutorials (also called the Stata YouTube channel). This library of videos covers a wide variety of topics, including Stata graphics. You can find these videos in a few different ways:

1. Visit the *Video Tutorials* webpage at `https://www.stata.com/links/video-tutorials/`, which lists all the video tutorials grouped by topics (including graphics).
2. Search the web for *StataCorp YouTube* to go to the *Stata YouTube Channel*; then click *PLAYLISTS* and then *Creating Graphs in Stata*.
3. You could search the Internet for *Stata YouTube Graphs*, which yields results from the *Stata YouTube Channel* as well as videos posted by others illustrating topics on Stata graphs.

I would especially recommend the following Stata Video Tutorials:

1. Basic scatterplots in Stata. This illustrates how you can use the point-and-click interface to create basic scatterplots in Stata and how you can use the point-and-click interface to apply modifications to the graph. It concludes by showing you the graph command that created the scatterplot, allowing you to then use that command to directly

create your customized scatterplot. Although this video might seem that it is only about creating scatterplots, it is also a great example of illustrating how you can use the Stata point-and-click interface to create a basic graph, iteratively use the point-and-click interface to customize it, and then save the command that created the customized graph.

2. Modifying sizes of elements in graphs. This illustrates how you can resize elements of your graphs interactively, allowing you to fiddle with the sizing until you find the size that you like best. Once you have found the right size, the video shows how you can replicate those changes via command-based options. Although this video refers to sizing of objects, you can use the same techniques with regard to other object characteristics. For example, you could play with the color of objects, interactively choosing different colors until you find the color you like best.
3. Modifying graphs using the Graph Editor. This video shows how you can *touch up* your graph using the Graph Editor, a point-and-click interface that allows you to modify your graph. It also shows how you can save those touch ups as a recording, which you can replay, allowing you to apply those touch ups in the future, even as part of a Stata graph command. This technique can be extremely useful if you want to make a change to your graph that is simpler via pointing and clicking, such as adding an arrow pointing at a particular observation or perhaps adding a text annotation at an exact place in the graph. You certainly can make such changes using command-based methods, but it can take a bit of trial and error to get the arrow or text placed in the exact position you wish. Instead, as this video illustrates, you can use the Graph Editor and Graph Recorder to create and record a touch up (like adding an arrow) and then replay that recording at a later time, applying that touch up to a graph. As the video illustrates, you can even replay the recording as part of your graph commands, allowing you to perform the touch ups within your do-files. For more information about using the Graph Editor, you can see `help graph editor`.

Even though this book focuses on using Stata commands for creating graphs, I hope that you find that these examples illustrate how the point-

and-click interface can be a useful tool for creating Stata graphs and for offering an interactive way to customize your graphs with the end goal of creating a command that performs your desired customizations.

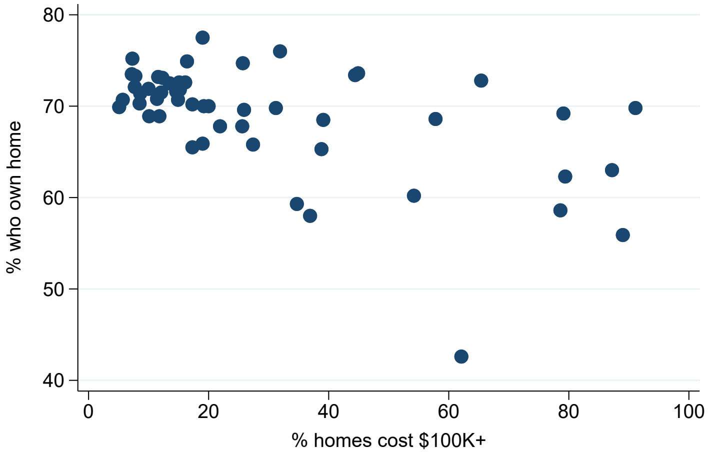
Chapter 2
Twoway graphs

The `graph twoway` command represents not just one kind of graph but actually over thirty different kinds of graphs. Many of these graphs are similar in appearance and function, so I have grouped them into nine families, which form the first nine sections of this chapter. These first nine sections, which discuss scatterplots to contour plots, cover the general features of these graphs and briefly mention some important options. The next section gives an overview of the options you can use with twoway graphs. [For further details about the options you can use with twoway graphs, see Options (section 8).] The chapter concludes with a section illustrating how you can overlay twoway graphs. For more details about twoway graphs, see `help graph twoway`.

2.1 Scatterplots

This section covers the use of scatterplots. Because scatterplots are so commonly used, this section will cover more details about the use of these graphs than subsequent sections. Also, this section introduces some of the options that we can use with many twoway plots, with cross-references to Options (section 8).

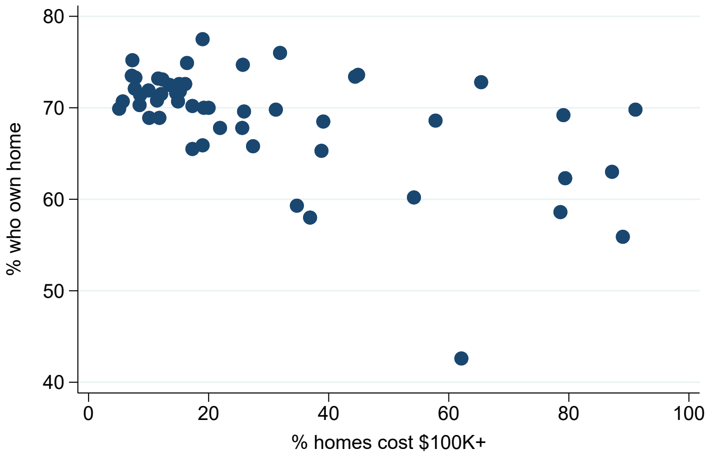
```
graph twoway scatter ownhome propval100
```



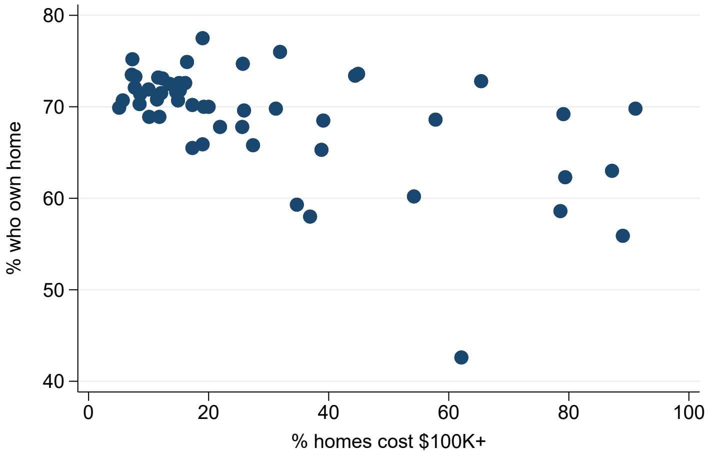
Here is a basic scatterplot. This command starts with `graph twoway`, which indicates that this is a twoway graph. `scatter` indicates that we are creating a twoway scatterplot. We next list the variables to be placed on the y and x axis, respectively. *Uses allstates.dta*

```
twoway scatter ownhome propval100
```

Because it can be cumbersome to type `graph twoway scatter`, Stata allows us to shorten this command to `twoway scatter`. *Uses allstates.dta*



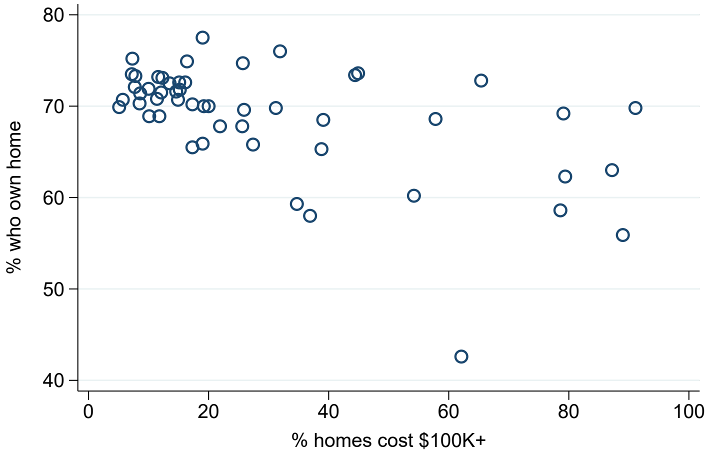
```
scatter ownhome propval100
```



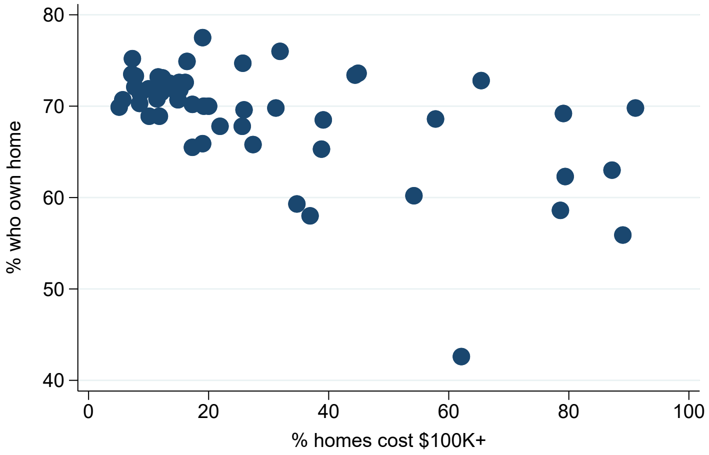
In fact, some `graph twoway` commands are so frequently used that Stata permits us to omit `graph twoway`. Here we start the command with `scatter`. Although this omission can save some typing, it can sometimes conceal the fact that the command is really a twoway graph and that these are a special class of graphs. For clarity, I will generally present these graphs starting with `twoway`. *Uses allstates.dta*

```
twoway scatter ownhome propval100, msymbol(Oh)
```

We can control the marker symbol with the `msymbol()` option. Here we make the symbols large, hollow circles. See Options : Markers (section 8.1) for more details about controlling the marker symbol, size, and color. See Styles : Symbols (section 10.11) for the available symbols. *Uses allstates.dta*



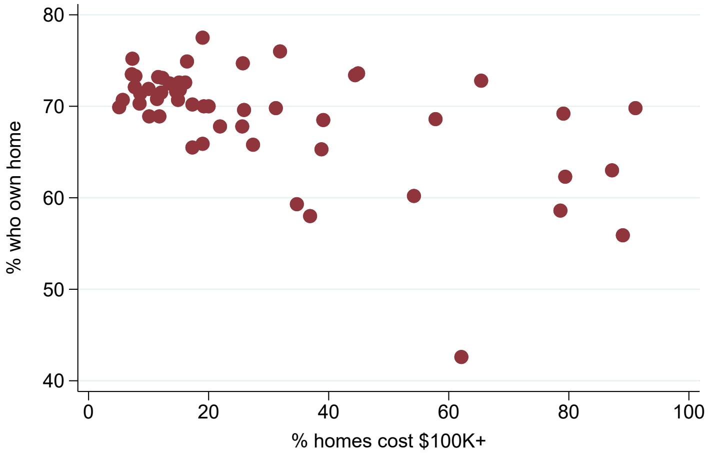
```
twoway scatter ownhome propval100, msize(vlarge)
```



We can control the marker size with the `msize()` option. By using `msize(vlarge)`, we make the markers very large. See **Styles : Markersize** (section 10.9) for other available sizes (including points, inches, and centimeters). Also see **Options : Markers** (section 8.1) for more details about markers. *Uses allstates.dta*

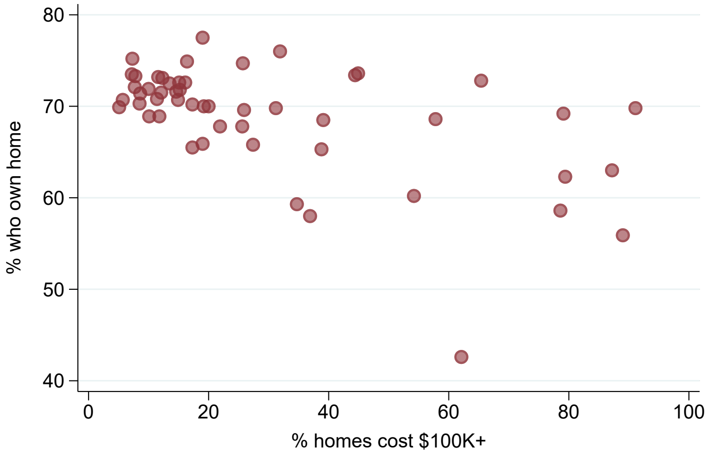
```
twoway scatter ownhome propval100, mcolor(maroon)
```

We can control the marker color with the `mcolor()` option. Using the `mcolor(maroon)`, we display the markers in `maroon`. See **Styles : Colors** (section 10.2) for more about colors you can choose, how to specify colors, color brightness (intensity), and transparency (opacity). *Uses allstates.dta*



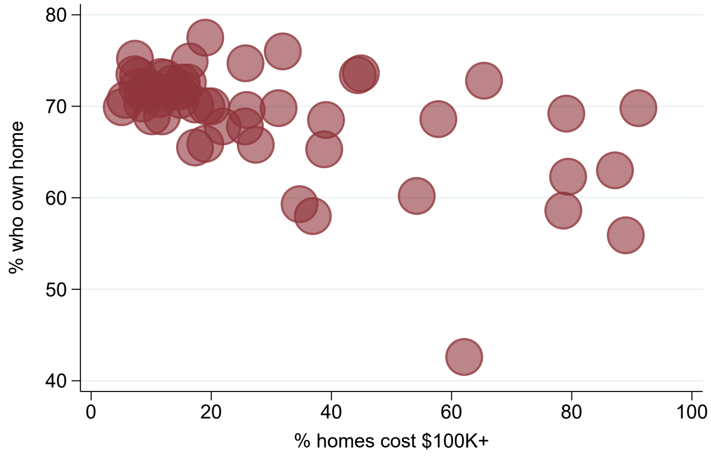
The following examples will illustrate ways of handling overlap when creating scatterplots. When observations overlap, this can conceal the quantity of observations with the overlapping values. This section will illustrate different methods of addressing such overlap, including modifying the opacity of markers. By reducing opacity, more observations drawn at the same location create overlap that is darker in color. For more information on opacity, see section 10.2.3.

```
twoway scatter ownhome propval100, mcolor(maroon%60)
```



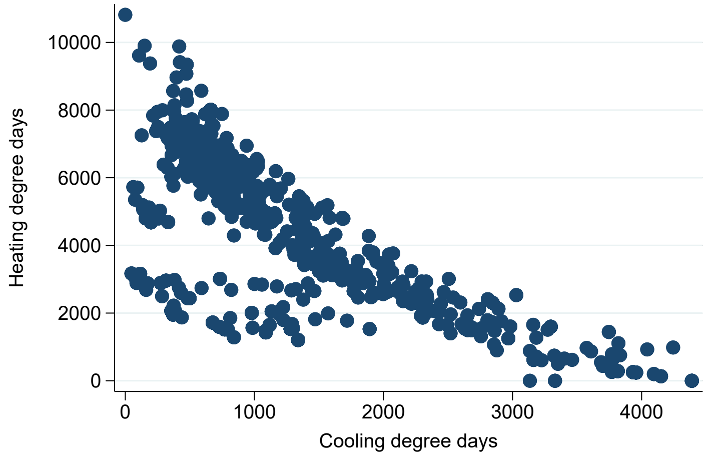
We can control the marker color with the `mcolor()` option. Here we change the marker color to `maroon%60`, specifying 60% opacity (that is, 40% transparency). See **Styles : Colors** (section 10.2) for more about colors you can choose and how to specify colors, color brightness (intensity), and transparency (opacity). *Uses allstates.dta*

```
twoway scatter ownhome propval100, mcolor(maroon%60)
msize(*3)
```



To help us see the impact of decreasing the opacity, we can increase the size of the markers using the `msize(*3)` option, increasing the marker size to three times their normal size. This also illustrates how decreased opacity can be useful if your graph contains large markers. *Uses allstates.dta*

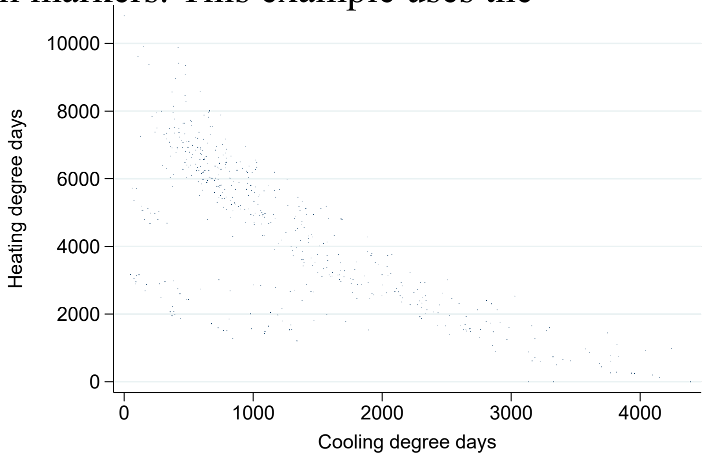
```
twoway scatter heatdd cooldd
```



The size and opacity of marker symbols can become especially important when you are graphing many overlapping data points. Consider this example that uses the `citytemp` dataset. Notice parts of the graph that have clumps of observations. The overlapping clumps are so saturated that it is unclear how many data points are located in those clumps. Let's try some other strategies for displaying the markers that might represent this relationship better. *Uses citytemp.dta*

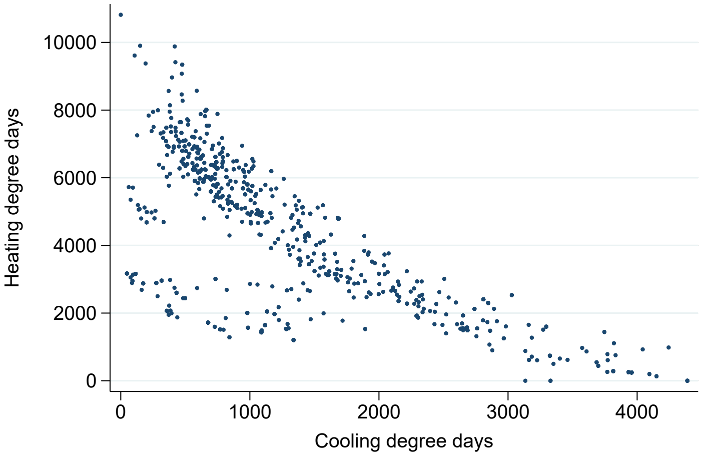
```
twoway scatter heatdd cooldd, msymbol(p)
```

One strategy is to use very small markers. This example uses the `msymbol(p)` option to display each marker using a tiny point. If those points are too small, you might want to use the `msize()` option to make the points bigger. But the `msize()` option does change the symbol size when using `msymbol(p)`. *Uses citytemp.dta*



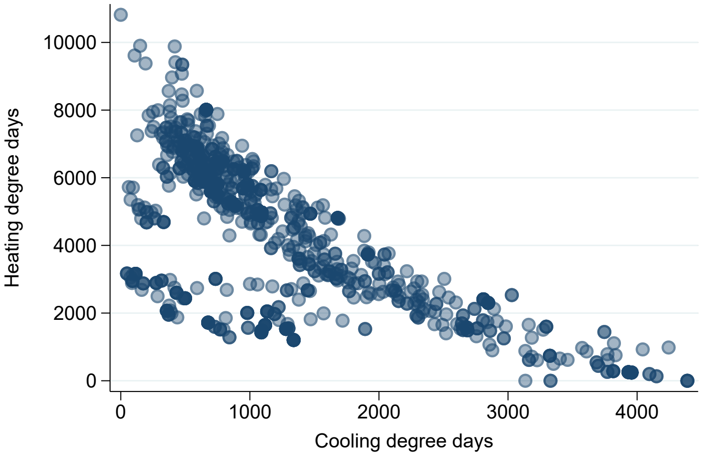
```
twoway scatter heatdd cooldd, msymbol(o) msize(vsmall)
```

To create larger markers (than the previous example), we use the `msymbol(o)` option to specify small circles as the marker and the `msize(vsmall)` option to size each marker as very small. See **Styles : Symbols** (section 10.11) and **Styles : Markersize** (section 10.9) for other symbols and sizes you could choose. *Uses citytemp.dta*



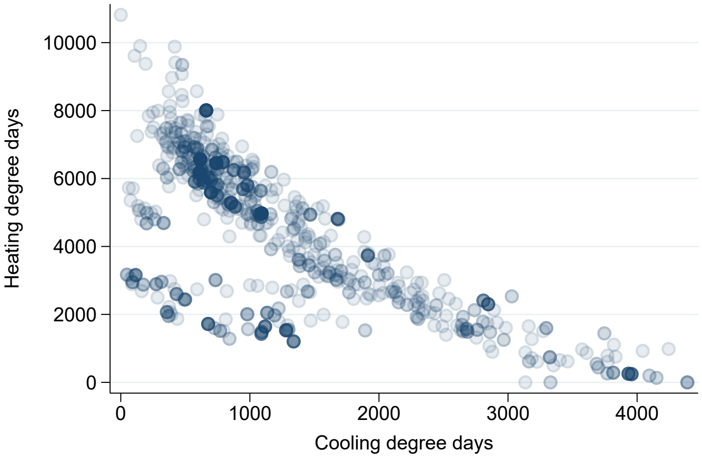
```
twoway scatter heatdd cooldd, mcolor(%40)
```

Another approach for handling numerous overlapping data points is to reduce opacity for the markers. This example specifies `mcolor(%40)`, displaying markers with their default color, but with 40% opacity. The darkness of the overlapping observations reflects the number of observations located in that part of the graph. *Uses citytemp.dta*



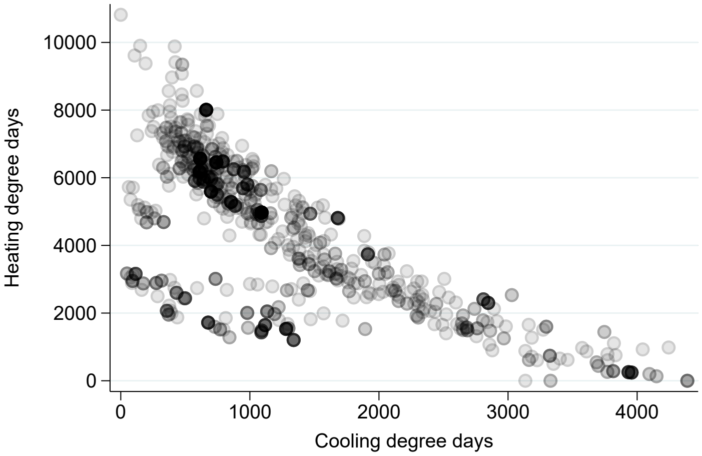
```
twoway scatter heatdd cooldd, mcolor(%10)
```

In the prior example are many places where the scatterplot is saturated with observations. This example shows an even lower level of opacity, specifying `mcolor(%10)` to draw each marker with only 10% opacity. *Uses citytemp.dta*

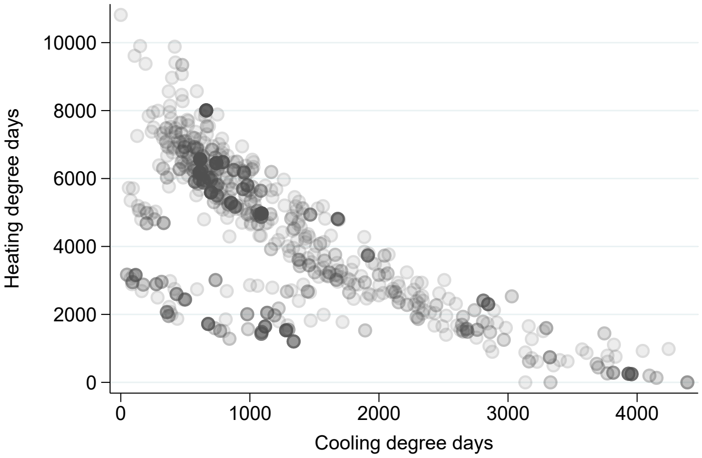


```
twoway scatter heatdd cooldd, mcolor(black%10)
```

You might want the markers in black for printing or a publication. This example changes the color to black but uses the same opacity as the prior example by using the `mcolor(black%10)` option. *Uses citytemp.dta*



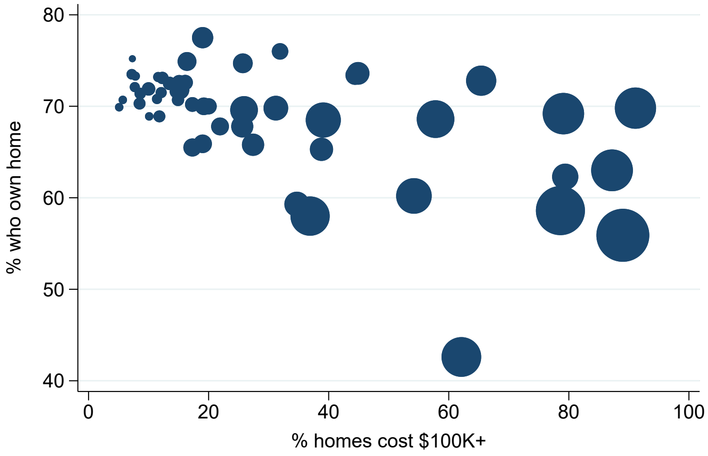
```
twoway scatter heatdd cooldd, mcolor(gs5%10)
```



This example is like the prior example but instead uses a shade of gray instead of black by specifying `mcolor(gs5%10)`. You can see Styles : Colors (section 10.2) for more about the colors you can choose, including how to specify different shades of gray. *Uses citytemp.dta*

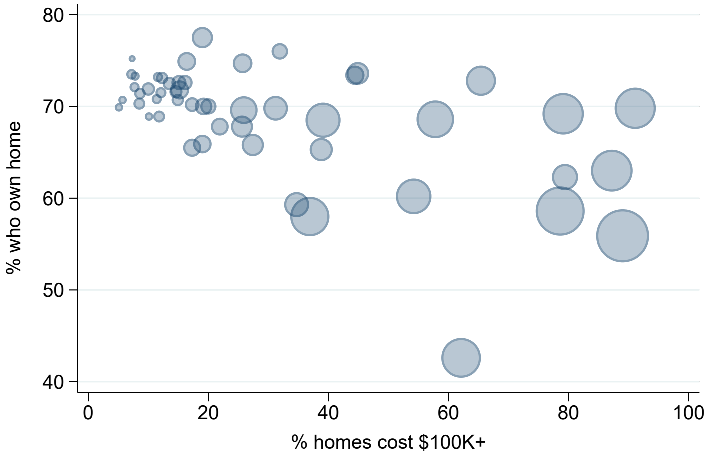
```
twoway scatter ownhome propval100 [aweight=rent700],
msize(small)
```

Returning back to `allstates.dta`, we see that this example uses a weight variable to determine the size of the symbols. By using `[aweight=rent700]`, we size the symbols according to the proportion of rents that exceed $700 per month, allowing us to graph three variables at once. We add the `msize(small)` option to shrink the size of all the markers so they do not get too large. See Options : Markers (section 8.1) for more details. *Uses allstates.dta*



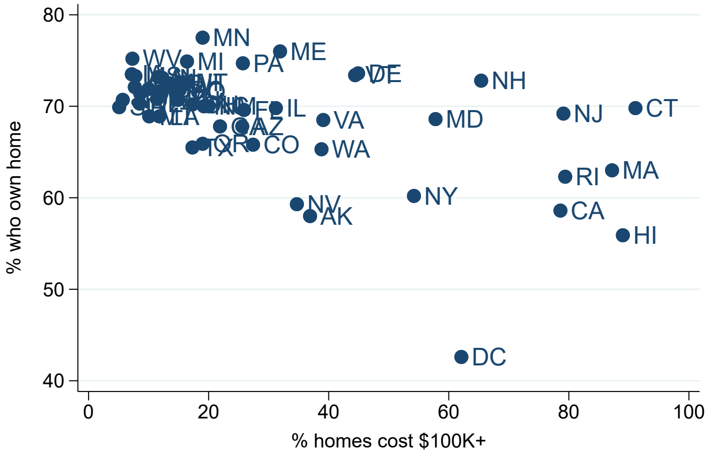
```
twoway scatter ownhome propval100 [aweight=rent700],
msize(small) mcolor(%30)
```

We repeat the prior example, adding the `mcolor(%30)` option to display the markers with 30% opacity. I prefer this graph to the prior one. When large markers are displayed, I prefer the way that they look using reduced opacity. *Uses allstates.dta*



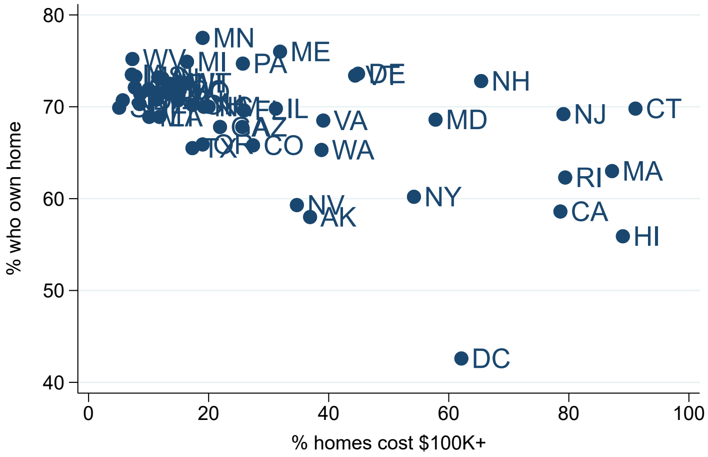
```
twoway scatter ownhome propval100, mlabel(stateab)
```

The `mlabel(stateab)` option adds a marker label with the state abbreviation. See Options : Marker labels (section 8.2) for more details about controlling the size, position, color, and angle of marker labels. *Uses allstates.dta*



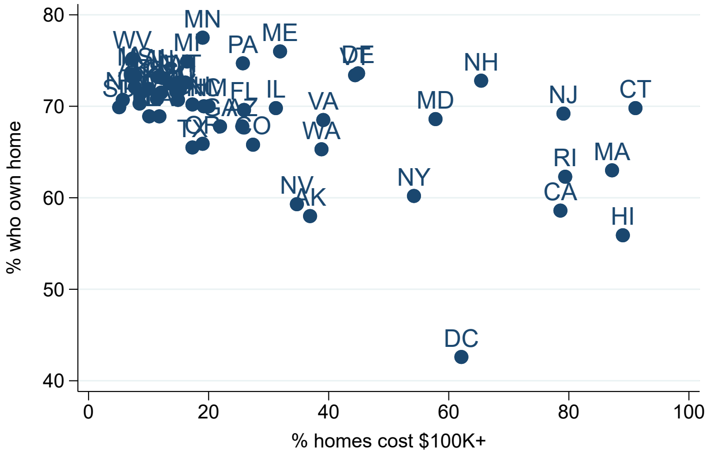
```
twoway scatter ownhome propval100, mlabel(stateab)
mlabsize(vlarge)
```

The `mlabsize()` option controls the marker label size. Here we make the marker label very large. *Uses allstates.dta*



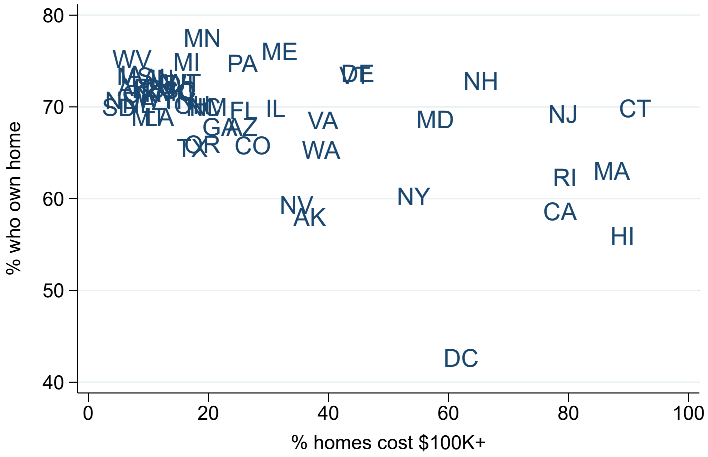
```
twoway scatter ownhome propval100, mlabel(stateab)
mlabposition(12)
```

The `mlabposition()` option controls the marker label position with respect to the marker. Here we place the marker labels at the twelve o’clock position, directly above the markers. See Options : Marker labels (section 8.2) for more examples. *Uses allstates.dta*



```
twoway scatter ownhome propval100, mlabel(stateab)
mlabposition(0) msymbol(i)
```

The `mlabposition(0)` option places the marker label in the center. The `msymbol(i)` option makes the marker symbol invisible. This replaces the marker symbols with the marker labels. *Uses allstates.dta*

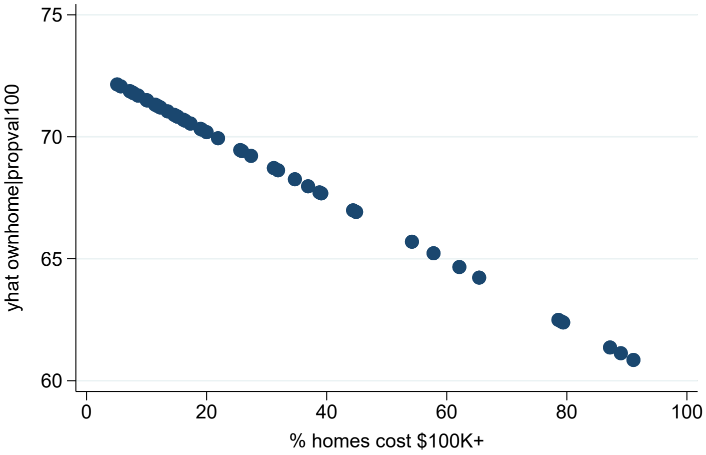


```
twoway scatter fv propval100
```

Say that we ran the following commands:

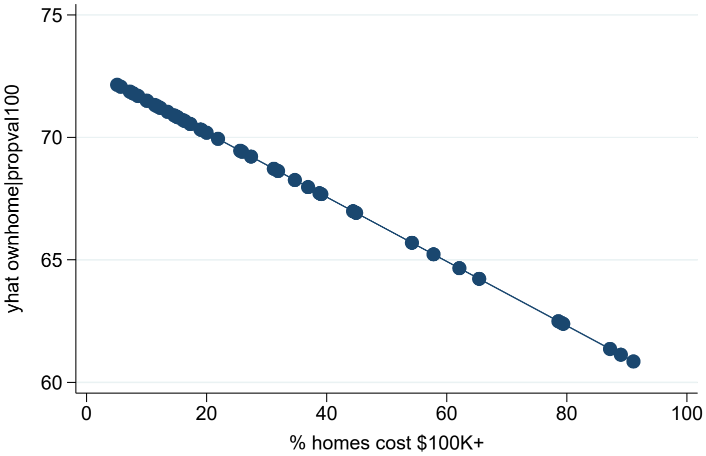
```
. regress ownhome propval100
. predict fv
```

The variable `fv` represents the fit values, and here we graph `fv` against `propval100`. As we expect, all the points fall along a line, but they are not connected. The next few examples will discuss options we can use to connect points; see Options : Connecting (section 8.3) for more details. *Uses allstates.dta*



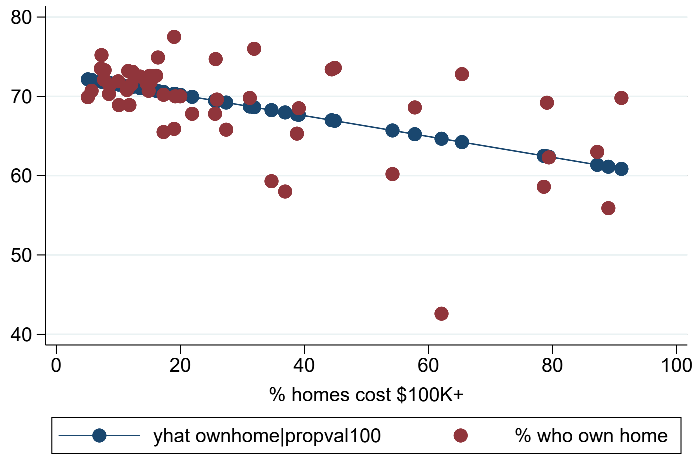
```
twoway scatter fv propval100, connect(l) sort
```

We add the `connect(l)` option to indicate that the points should be connected with a line. We also add the `sort` option, which is generally recommended when we connect observations and the data are not already sorted on the x variable. *Uses allstates.dta*



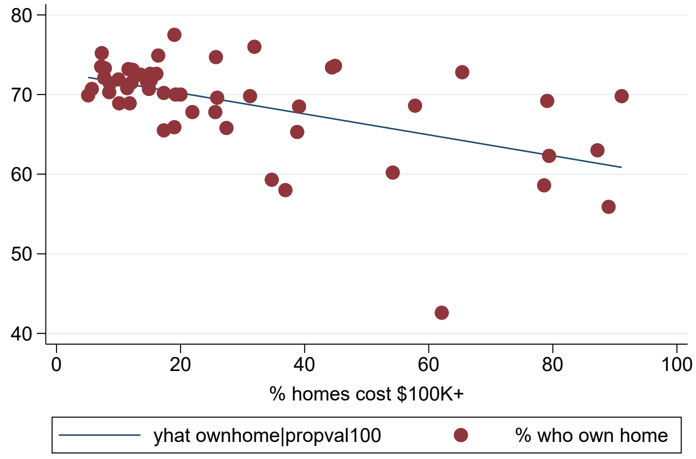
```
twoway scatter fv ownhome propval100, connect(l i) sort
```

We can show both the observations and the fit values in one graph. The `connect(l i)` option specifies that the first y variable should be connected with straight lines (`l` for line) and the second y variable should not be connected (`i` for invisible connection). *Uses allstates.dta*



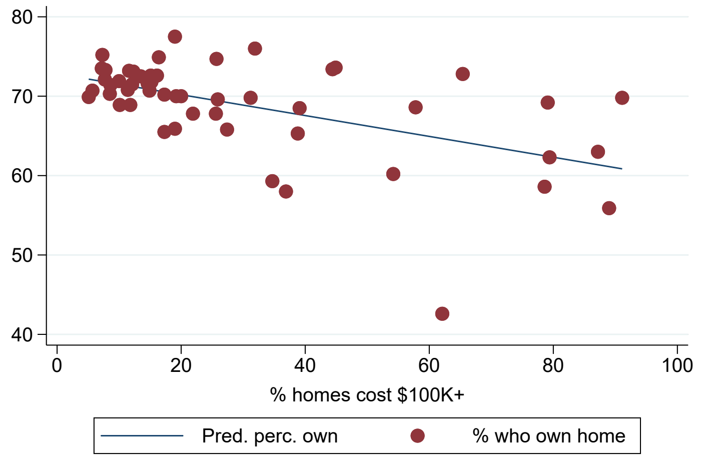
```
twoway scatter fv ownhome propval100, msymbol(i .) connect(l
i) sort
```

The `msymbol(i .)` option specifies that the first y variable should not have symbols displayed (`i` for invisible symbol) and that the second y variable should have the default symbols displayed. *Uses allstates.dta*



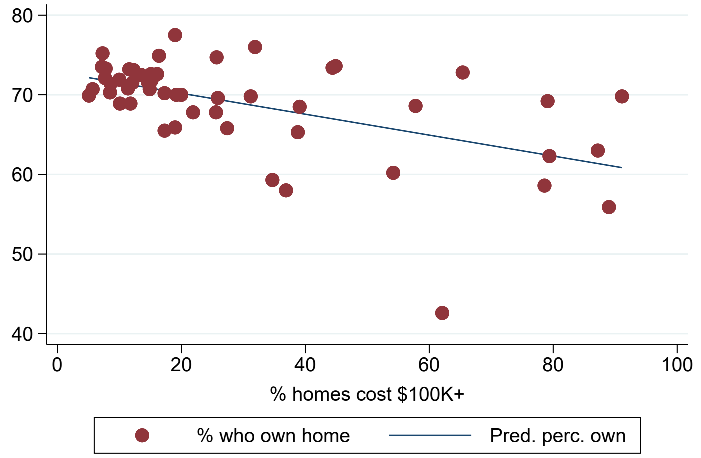
```
twoway scatter fv ownhome propval100, msymbol(i .) connect(l
i) sort legend(label(1 Pred. perc. own))
```

The `legend()` option controls the legend. We use `label()` within the `legend()` option to specify the contents of the first item in the legend. See Options : Legend (section 8.9) for more details on the legend. *Uses allstates.dta*



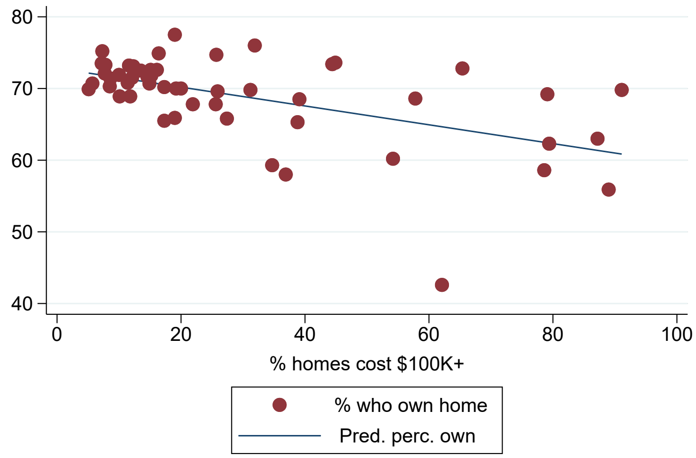
```
twoway scatter fv ownhome propval100, msymbol(i .) connect(l
i) sort legend(label(1 Pred. perc. own) order(2 1))
```

The `order()` option within the `legend()` option specifies the order in which the items in the legend are displayed. *Uses allstates.dta*



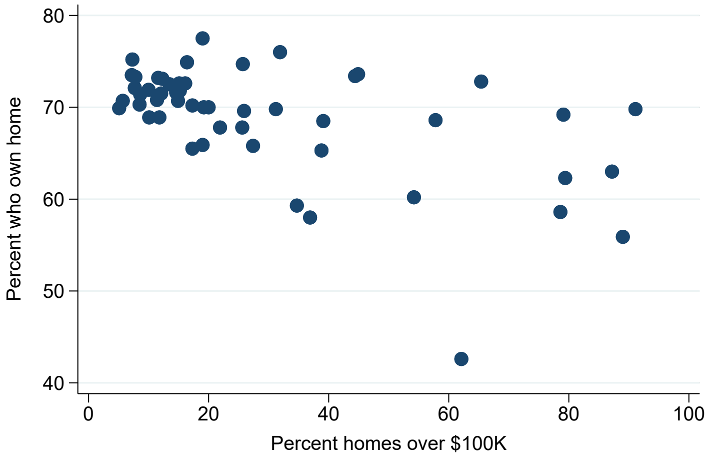
```
twoway scatter fv ownhome propval100, msymbol(i .) connect(l
i) sort legend(label(1 Pred. perc. own) order(2 1) cols(1))
```

The `cols(1)` option makes the items in the legend display in one column. *Uses allstates.dta*



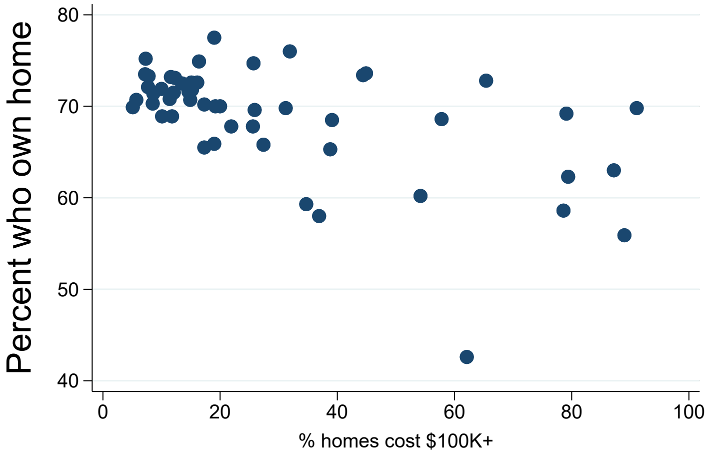
```
twoway scatter ownhome propval100, xtitle("Percent homes over
$100K") ytitle("Percent who own home")
```

The `xtitle()` and `ytitle()` options specify the titles for the x and y axes. See Options : Axis titles (section [8.4](#)) for more details about how to control the display of axes. *Uses allstates.dta*

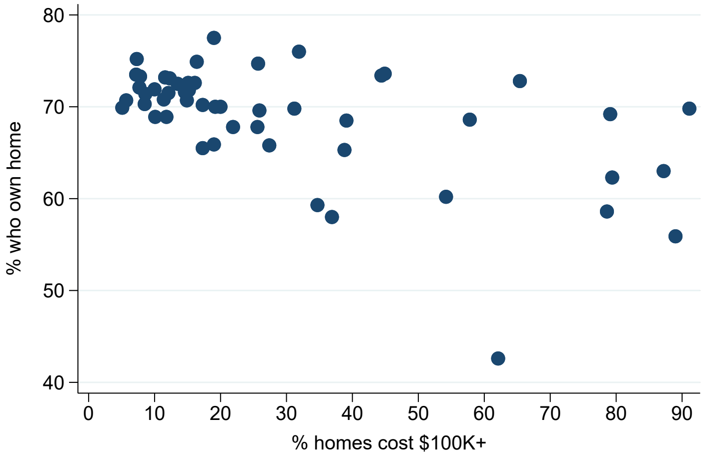


```
twoway scatter ownhome propval100, ytitle("Percent who own
home", size(huge))
```

Here we use the `size(huge)` option to make the title on the y axis huge. For other text sizes—including points, inches, and centimeters—see Styles : Textsize (section [10.12](#)). *Uses allstates.dta*



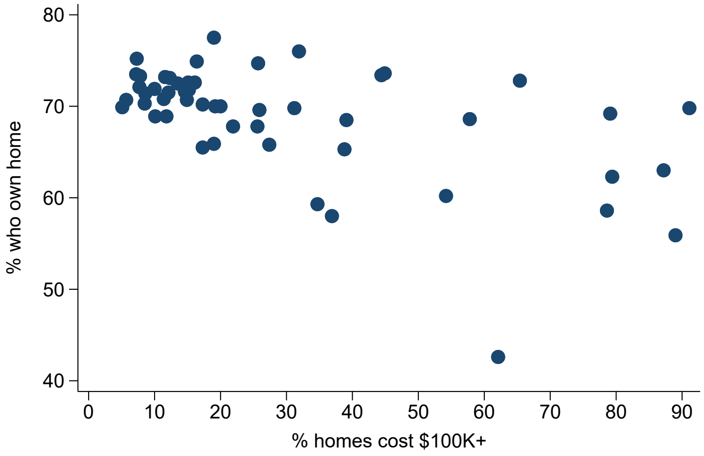
```
twoway scatter ownhome propval100, xlabel(#10) ylabel(#5)
```



In this example, we use the `xlabel(#10)` option to ask Stata to use approximately 10 nice labels and the `ylabel(#5)` option to use approximately 5 nice labels. Here our gentle request was observed exactly, but sometimes Stata will choose somewhat different values to create axis labels it believes are logical. See Options : Axis labels (section 8.5) for more details on labeling axes. *Uses allstates.dta*

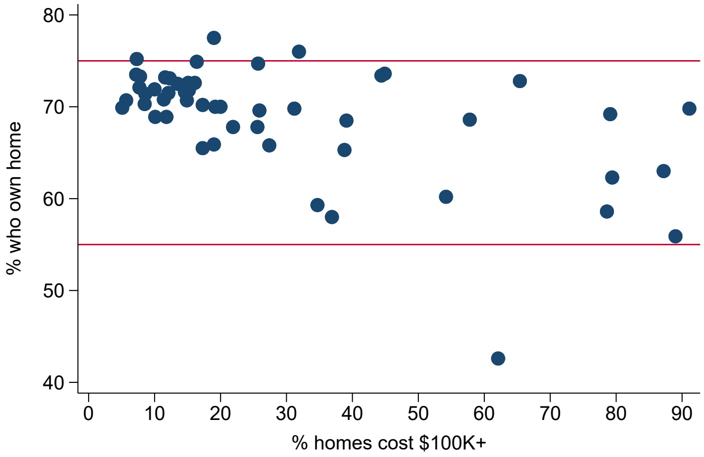
```
twoway scatter ownhome propval100, xlabel(#10) ylabel(#5,
nogrid)
```

We use the `nogrid` option within the `ylabel()` option to suppress the grid for the y axis (and we could show the grid by adding the `grid` option). You can also specify `grid` or `nogrid` within the `xlabel()` option to control grids for the x axis. For more details, see Options : Axis labels (section 8.5). *Uses allstates.dta*



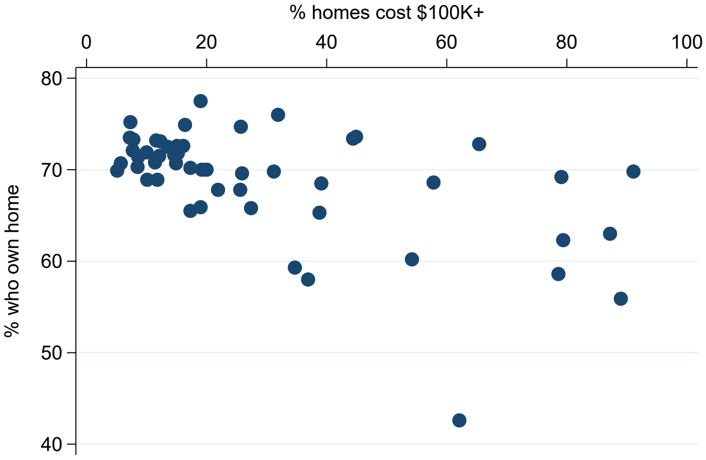
```
twoway scatter ownhome propval100, xlabel(#10) ylabel(#5,
nogrid) yline(55 75)
```

The `yline()` option adds a horizontal reference line where y equals 55 and 75. You can add a vertical reference line with the `xline()` option. *Uses allstates.dta*

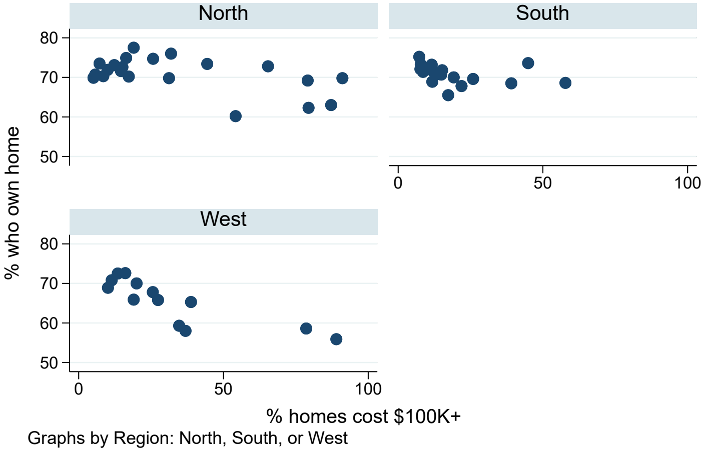


```
twoway scatter ownhome propval100, xscale(alt)
```

Here we use the `xscale()` option to request that the x axis be placed in its alternate position, which is at the top instead of at the bottom. To learn more about axis scales, including suppressing, extending, or relocating them, see Options : Axis scales (section 8.6). *Uses allstates.dta*



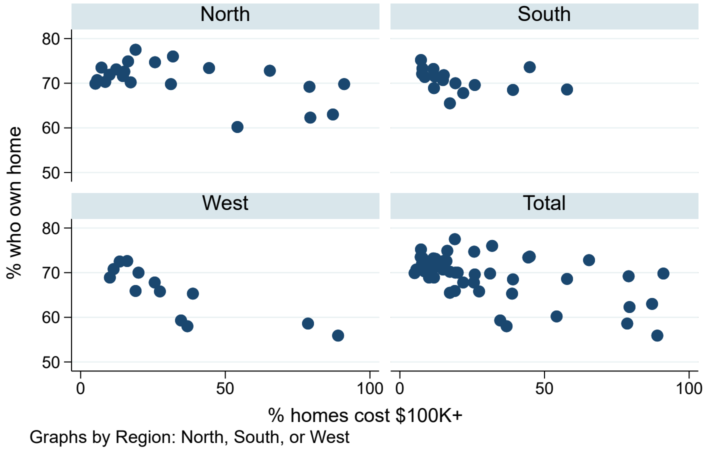
```
twoway scatter ownhome propval100, by(nsw)
```



We use the `by(nsw)` option here to make separate graphs for states in the North, South, and West. At the bottom left corner, there is a note that describes how the graphs are separated, which is based on the variable label for `nsw`. If this variable had not been labeled, the variable name would have used, that is, *Graphs by nsw*. See Options : By (section 8.8) for more details about using the `by()` option. *Uses allstates.dta*

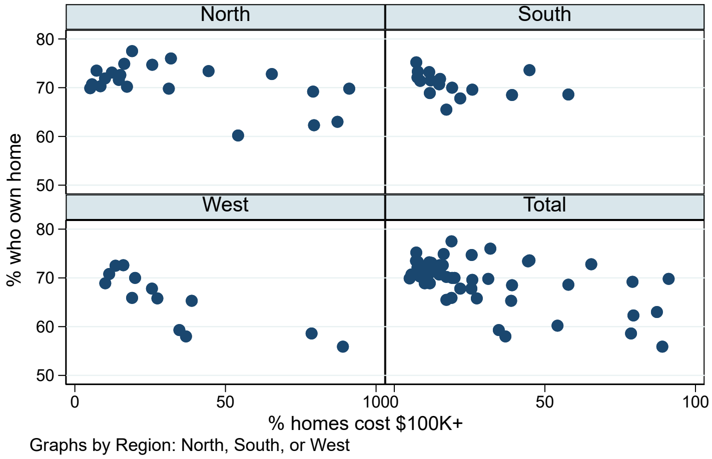
```
twoway scatter ownhome propval100, by(nsw, total)
```

We can use the `total` option within the `by()` option to add another graph showing all the observations. *Uses allstates.dta*



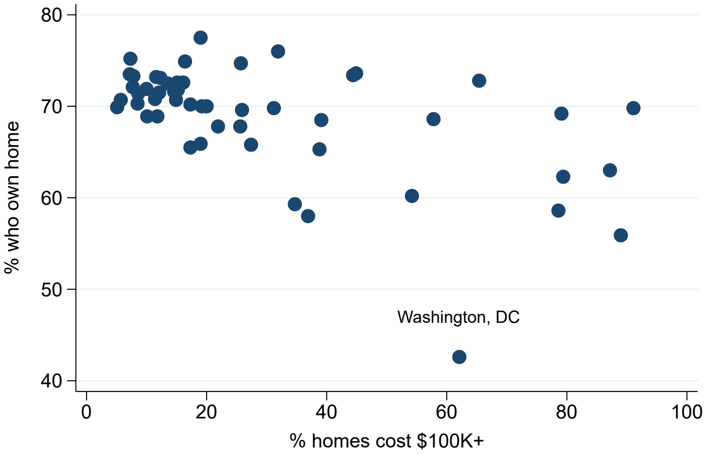
```
twoway scatter ownhome propval100, by(nsw, total compact)
```

The `compact` option within the `by()` option makes the graphs display more compactly. *Uses allstates.dta*



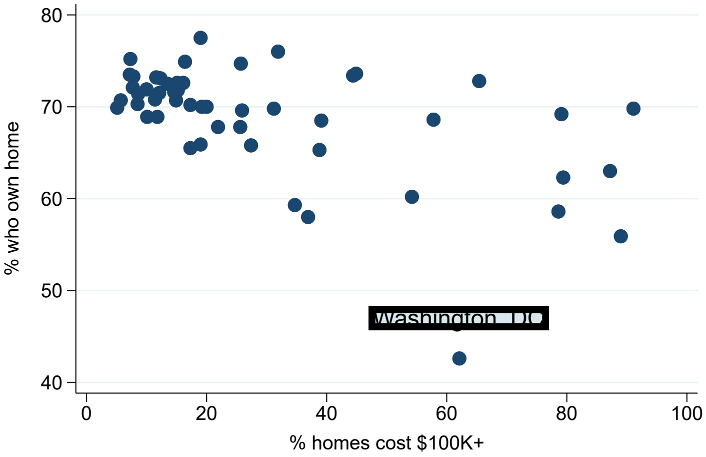
```
twoway scatter ownhome propval100, text(47 62 "Washington,
DC")
```

The `text()` option adds text to the graph. Here we add text to label the observation belonging to Washington, DC. See Options : Adding text (section 8.10) for more information about adding text. *Uses allstates.dta*



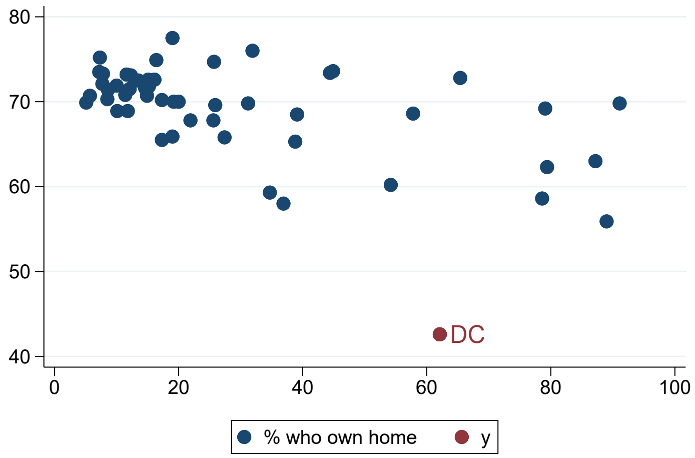
```
twoway scatter ownhome propval100, text(47 62 "Washington, DC", size(large) lwidth(vthick) box)
```

Here we make the text large and surround it with a thick-lined box. See Options : Textboxes (section 8.11) for more details. *Uses allstates.dta*

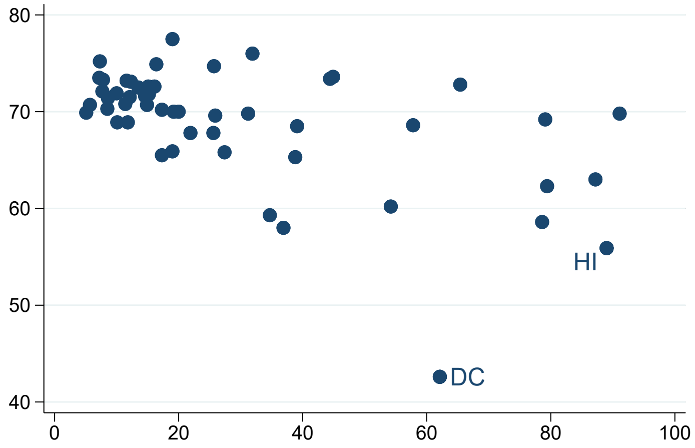


```
twoway (scatter ownhome propval100) (scatteri 42.6 62.1 "DC")
```

This graph uses the `scatteri` (scatter immediate) command to plot and label a point for Washington, DC. The values `42.6` and `62.1` are the values for `ownhome` and `propval100` for Washington, DC, and are followed by `"DC"`, which acts as a marker label for that point. *Uses allstates.dta*



```
twoway (scatter ownhome propval100) (scatteri 42.6 62.1 "DC"
55.9 89 (8) "HI"), legend(off) scheme(vg_samec)
```



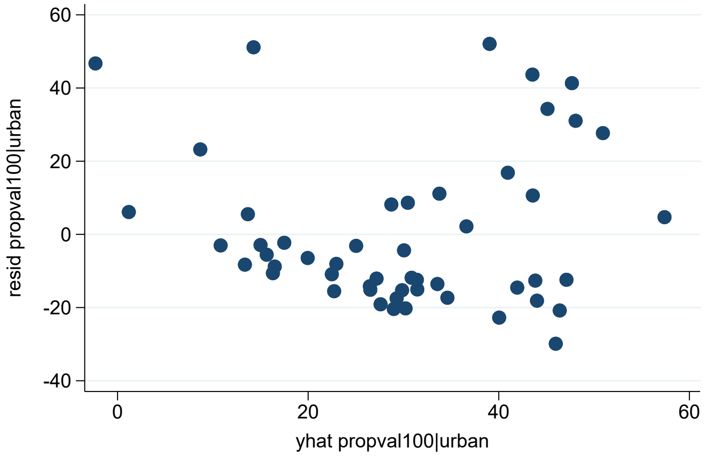
This graph also labels Hawaii at the eight o’clock position. The `legend(off)` option also suppresses the legend. Finally, this graph uses the `scheme(vg_samec)` option so the markers created with `scatteri` look identical to the other markers. *Uses allstates.dta*

Next let’s turn to more graph commands that make graphs similar to `twoway scatter`, namely, `twoway spike`, `twoway dropline`, and `twoway dot`. Most of the options that have been illustrated apply to these graphs as well, so they will not be repeated here.

```
twoway scatter r yhat
```

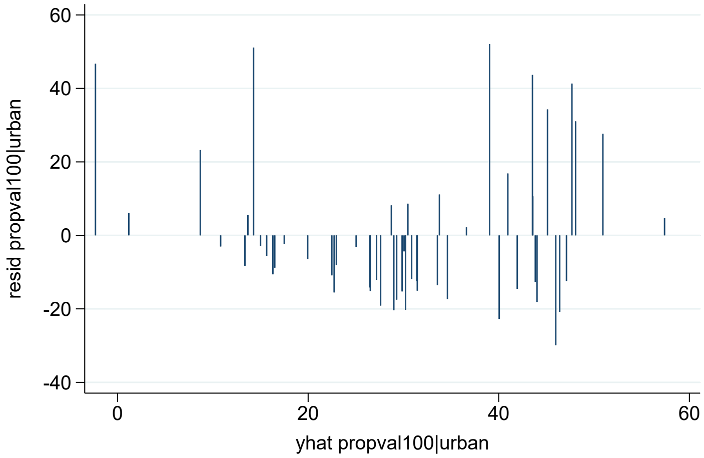
Imagine that we ran a regression predicting `propval100` from `urban` and generated the residual, calling it `r`, and the predicted value, calling it `yhat`.

Consider this graph using the `scatter` command to display the residual by the predicted value. *Uses allstates.dta*



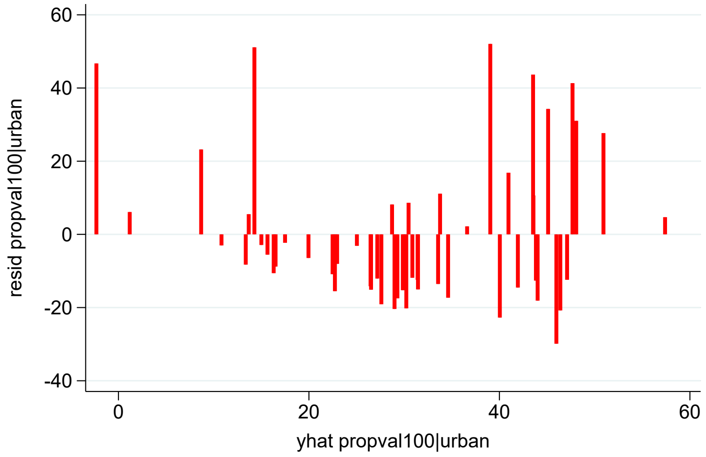
```
twoway spike r yhat
```

This same graph could be shown using the `spike` command. This command produces a spike plot, with each spike, by default, originating from 0. *Uses allstates.dta*



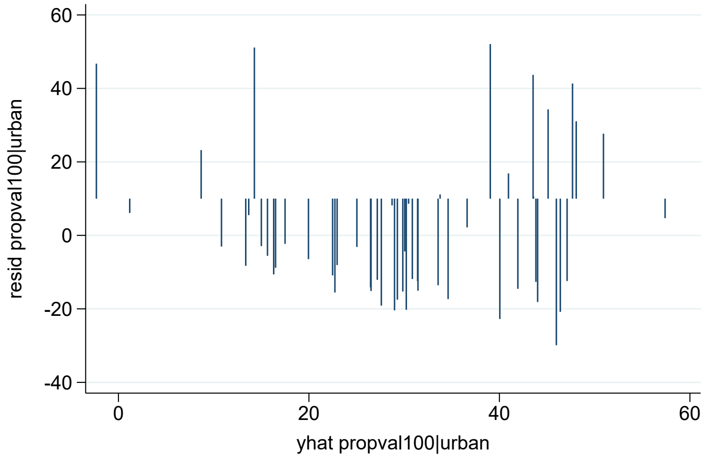
```
twoway spike r yhat, lcolor(red) lwidth(thick)
```

We can use the `lcolor()` (line color) option to set the color of the spikes and the `lwidth()` (line width) option to set the width of the spikes. Here we make the spikes thick and red. See Styles : Colors (section 10.2) for more details about specifying colors, and see Styles : Linewidth (section 10.7) for more details about specifying line widths. *Uses allstates.dta*



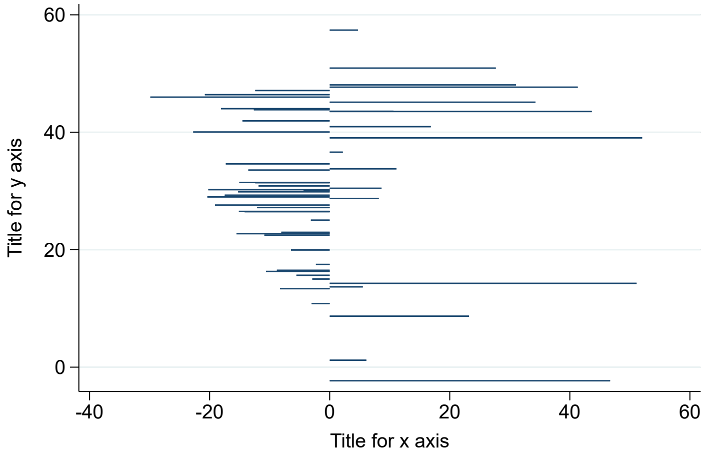
```
twoway spike r yhat, base(10)
```

By default, the base is placed at 0, which is a logical choice when displaying residuals because our interest is in deviations from 0. For illustration, we use the `base(10)` option to set the base of the y axis to 10, and the spikes are displayed with respect to 10. *Uses allstates.dta*



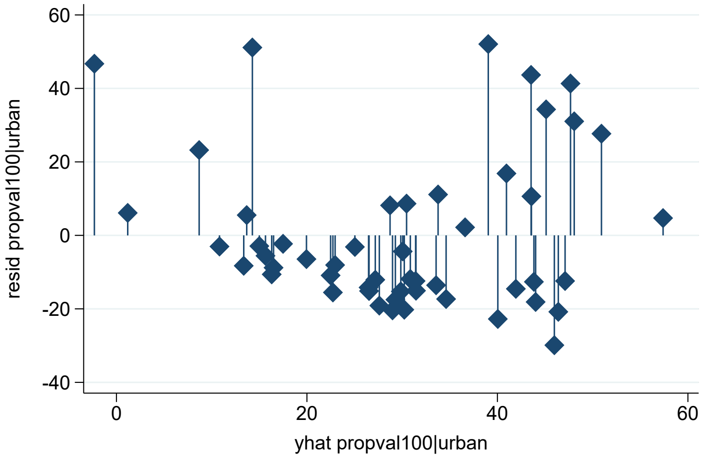
```
twoway spike r yhat, horizontal xtitle(Title for x axis)
ytitle(Title for y axis)
```

The `horizontal` option swaps the position of the `r` and `yhat` variables. The x axis remains at the bottom, and the y axis remains at the left. *Uses allstates.dta*



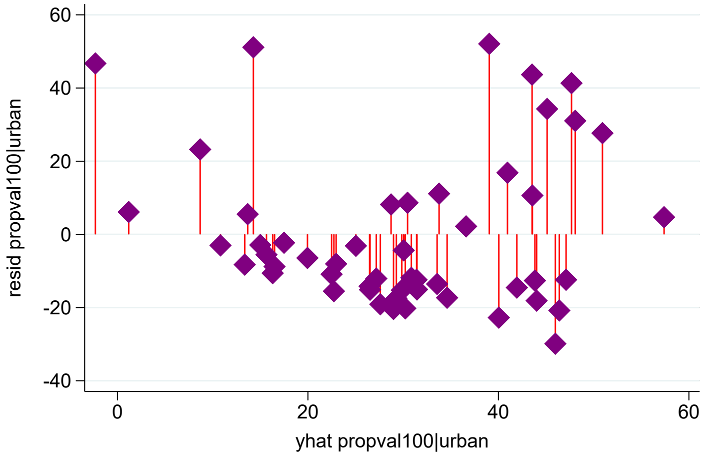
```
twoway dropline r yhat, msymbol(D)
```

A `twoway dropline` plot is much like a spike plot but permits a symbol, as well. It supports the `horizontal`, `base()`, `lcolor()`, and `lwidth()` options just like `twoway spike`. Here we add the `msymbol(D)` option to obtain diamonds as the symbols; see Options : Markers (section 8.1) for more details. *Uses allstates.dta*



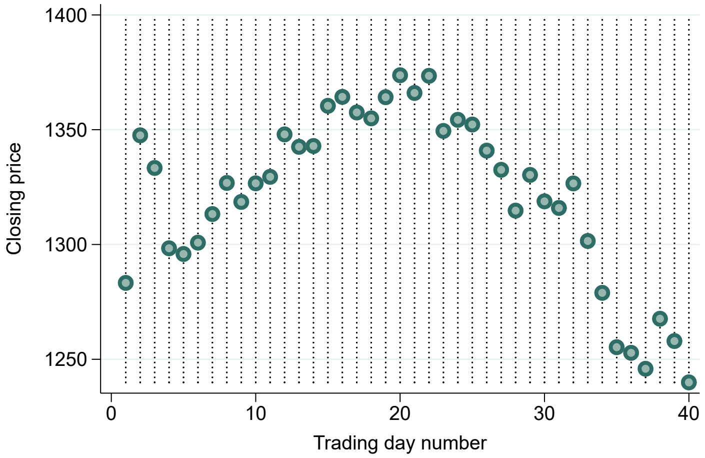
```
twoway dropline r yhat, msymbol(D) msize(large)
mcolor(purple) mlwidth(thick) lcolor(red)
```

Here we make the symbols large and purple and the lines thick and red. For more information, see Options : Markers (section 8.1). *Uses allstates.dta*



```
twoway dot close tradeday, msize(large) msymbol(O)
mfcolor(eltgreen) mlcolor(emerald) mlwidth(thick)
```

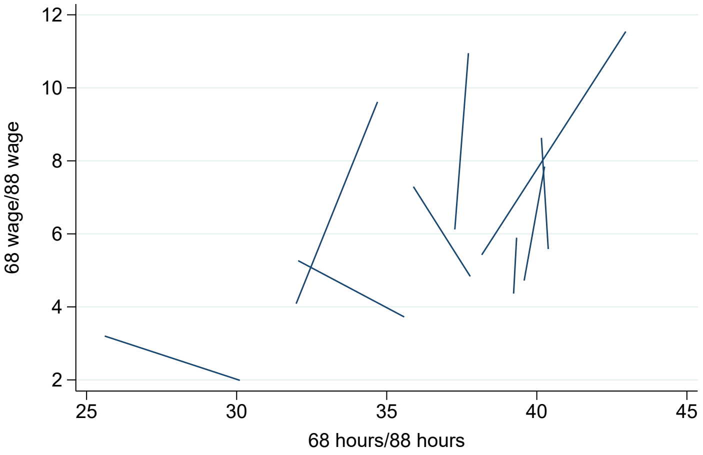
A `dot` plot is similar to a scatterplot but shows dotted lines for each x variable value, making it more useful when the x values are equally spaced. Here we look at the closing price of the S&P 500 by trading day and make the markers filled with eltgreen with thick emerald outlines.
Uses spjanfeb2001.dta



This section concludes with a special kind of scatterplot, a paired coordinate plot. As its name implies, this plot takes as its input a pair of coordinates, for example, (y_1, x_1) and (y_2, x_2), and then plots a line between each coordinate pair. The line may be plain (if you use `pcspike`), capped with symbols (if you use `pccapsym`), terminated with an arrow (if you use `pcarrow`), capped with bidirectional arrows (if you use `pcbarrow`), or a pair of marker symbols (if you use `pcscatter`). To illustrate these graphs, we will begin by using one of the built-in data files from Stata named `nlswide1`, which contains nine observations with aggregate data on nine different occupation classes for 1968 and for 1988. We will focus on hourly wages (`wage68` and `wage88`) and hours worked per week (`hours68` and `hours88`). These paired coordinate graphs will help us see how these two variables have changed over this 20-year span for these nine occupations.

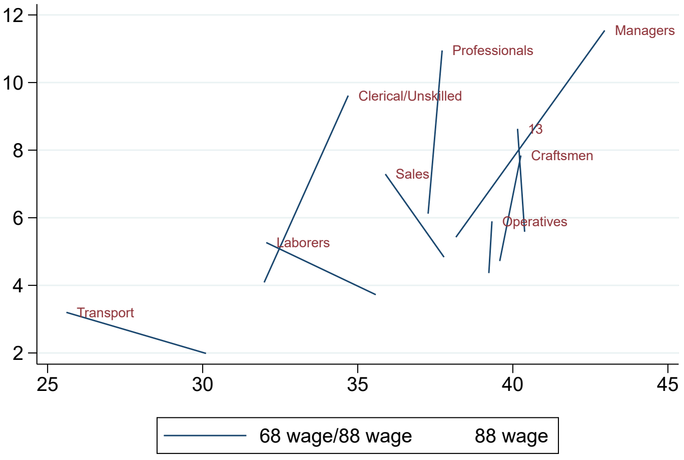
```
twoway (pcspike wage68 hours68 wage88 hours88)
```

The command's syntax is to provide the (y_1, x_1) variables for time 1 and then the (y_2, x_2) for time 2. This graph is interesting, but without any labels it is hard to understand the trends being conveyed. *Uses nlswide1.dta*



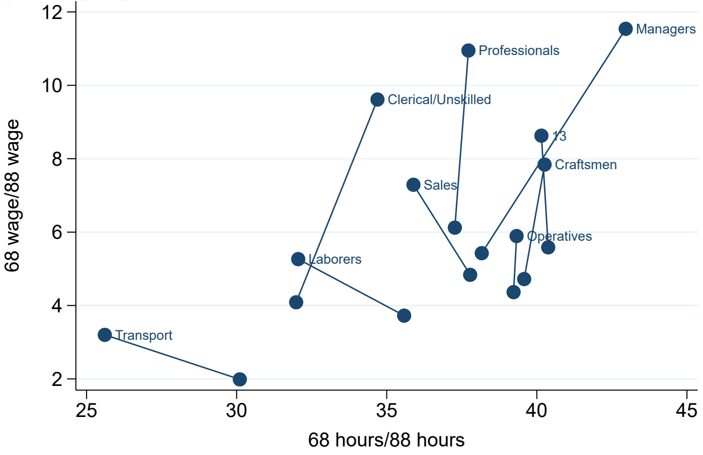
```
twoway (pcspike wage68 hours68 wage88 hours88) (scatter
wage88 hours88, msymbol(i) mlabel(occ) mlabsize(small))
```

We can combine the previous command with a `scatter` command to label the data in 1988 with the names of the occupations. This graph is easier to interpret. *Uses nlswide1.dta*



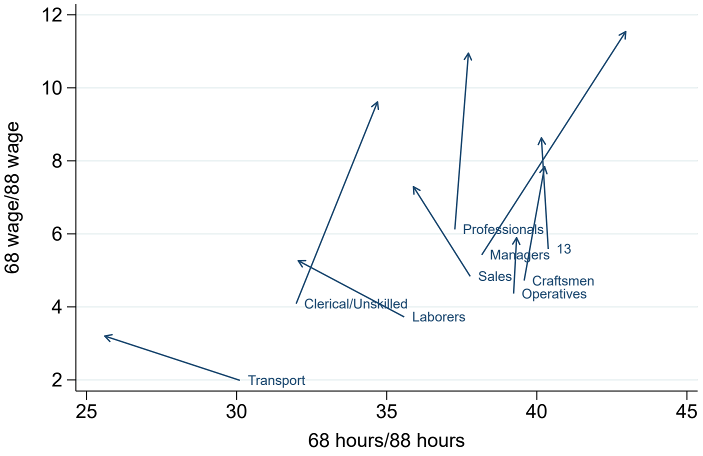
```
twoway pccapsym wage68 hours68 wage88 hours88, mlabel(occ)
mlabsize(small) headlabel
```

A simpler solution is to use the `pccapsym` command with the `mlabel()` option to label the lines with the occupation names. The `headlabel` option labels the head of the line (that is, 1988) rather than the default of labeling the tail (that is, 1968). A related command is `pcscatter` (not illustrated), which would display the symbols without the lines. *Uses nlswide1.dta*



```
twoway pcarrow wage68 hours68 wage88 hours88, mlabel(occ)
mlabsize(small)
```

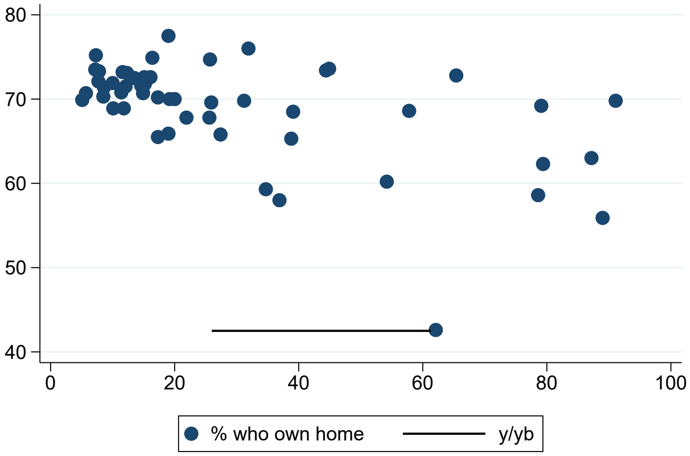
Because these data represent changes in time, the graph may be more compelling by displaying the lines as arrows. We now omit the `headlabel` option so the occupation names are displayed at the tail of the arrow. A related command is `pbcarrow` (not illustrated), which would display bidirectional arrows. *Uses nlswide1.dta*



In addition to the traditional forms of these commands, there are two immediate forms of these commands that permit us to supply coordinates as part of the command. The `twoway pci` command displays lines, whereas the `twoway pcarrowi` command displays arrows. We now switch to the `allstates` data file. We use an outlier in a scatterplot, with lines and arrows to call attention to the outlier.

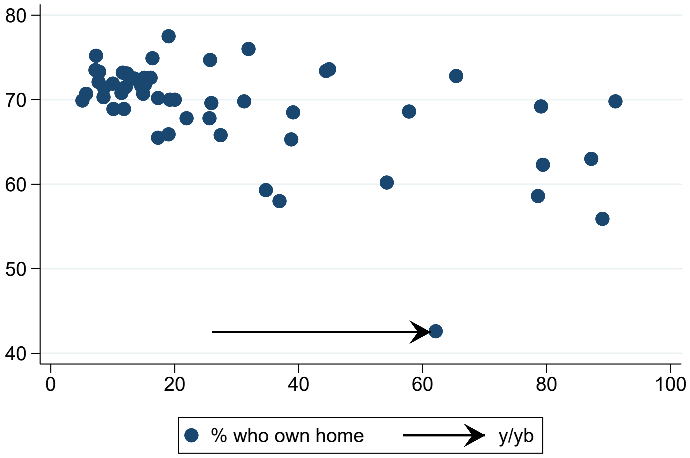
```
graph twoway (scatter ownhome propval100) (pci 42.5 26 42.5
61.3, lwidth(medthick) lcolor(black))
```

Here we add a line from (42.5,26) to (42.5,61.3) to help call attention to this outlying observation. However, an arrow might be more effective. *Uses allstates.dta*



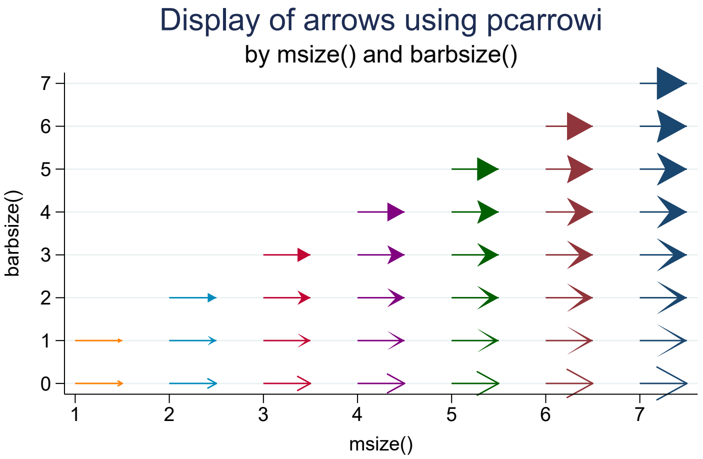
```
graph twoway (scatter ownhome propval100) (pcarrowi 42.5 26
42.5 61.3, lwidth(medthick) lcolor(black) msize(5)
barbsize(3) mcolor(black))
```

By using `pcarrowi`, we create an arrow with an arrowhead of size 5 and barb of size 3 that points to this outlying observation. *Uses allstates.dta*



```
no command
```

Here is a display of arrows using `pcarrowi` to illustrate the effect of `msize()` and `barbsize()`. The values of `msize()` range from 1 to 7, and the values of `barbsize()` range from 0 to 7. As you move to the right, the overall arrowhead size gets larger (due to increasing `msize()` from 1 to 7). As you move up, the amount of the arrowhead that is filled up increases (due to increasing `barbsize` from 0 to 7). *Uses none.dta*

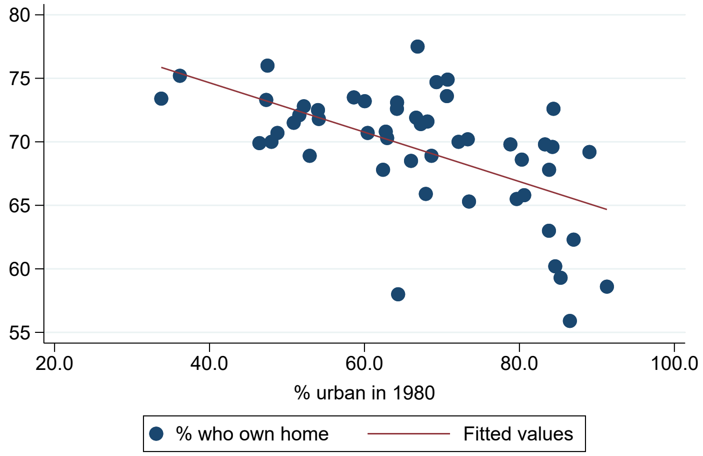


2.2 Regression fits and splines

This section focuses on the twoway commands you can use for displaying fit values: `lfit`, `qfit`, `fpfit`, `mband`, `mspline`, and `lowess`. For more information, see `help graph twoway lfit`, `help graph twoway qfit`, `help graph twoway fpfit`, `help graph twoway mband`, `help graph twoway mspline`, and `help graph twoway lowess`. We use the `allstates` data file, omitting Washington, DC.

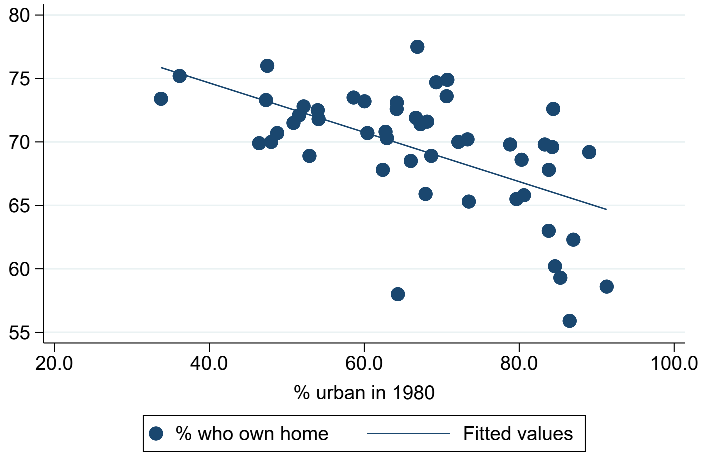
```
twoway (scatter ownhome pcturban80) (lfit ownhome pcturban80)
```

Here we show a scatterplot of `ownhome` by `pcturban80`. We also overlay a linear fit (`lfit`) predicting `ownhome` from `pcturban80`. See Twoway : Overlaying (section 2.11) for more information about overlaying twoway graphs. *Uses allstatesdc.dta*



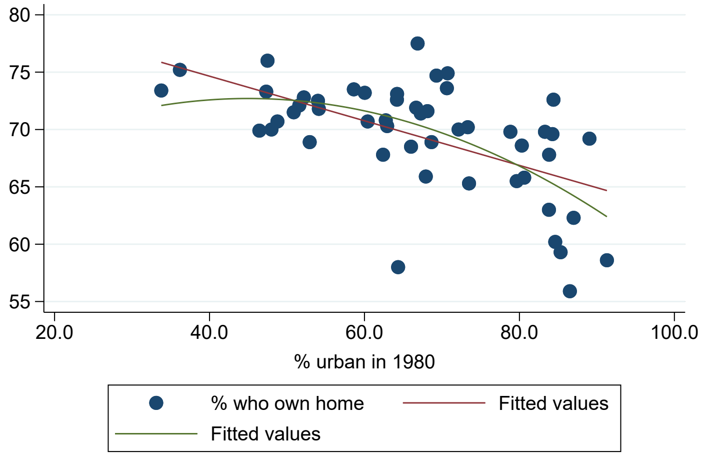
```
twoway (scatter ownhome pcturban80) (lfit ownhome
pcturban80), pcycle(1)
```

In the previous graph, the line color was different from the marker color. By adding the `pcycle(1)` option, we indicate that the plot characteristics should repeat after one iteration. So the scatterplot shows as blue markers, and then the line is shown as a matching blue line. *Uses allstatesdc.dta*



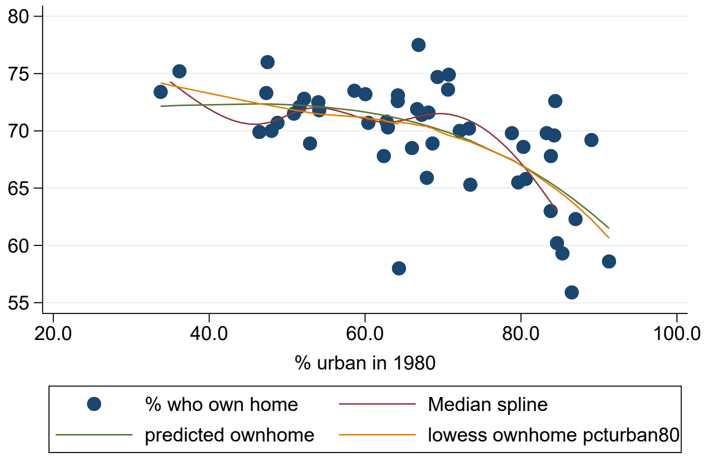
```
twoway (scatter ownhome pcturban80) (lfit ownhome pcturban80)
(qfit ownhome pcturban80)
```

It is sometimes useful to overlay fit plots to compare the fit values. Here we overlay a linear fit (`lfit`) and quadratic fit (`qfit`) and can see some discrepancies between them. *Uses allstatesdc.dta*



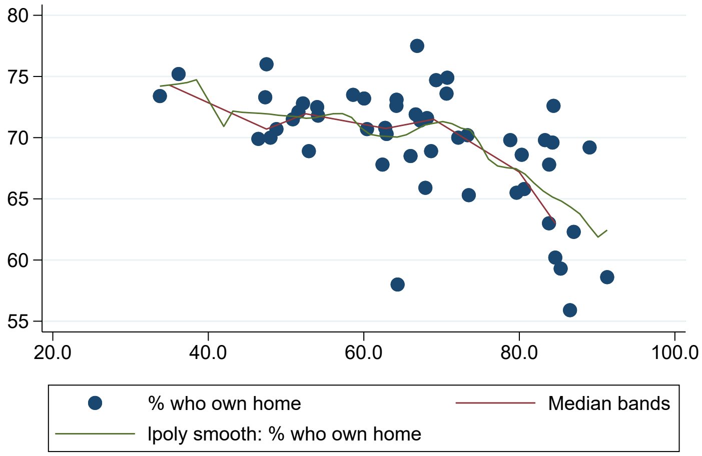
```
twoway (scatter ownhome pcturban80) (mspline ownhome
pcturban80) (fpfit ownhome pcturban80) (lowess ownhome
pcturban80)
```

Stata supports several other fit methods. Here we show a median spline (`mspline`) overlaid with a fractional polynomial fit (`fpfit`) and a locally weighted scatterplot smoothing (`lowess`). *Uses allstatesdc.dta*



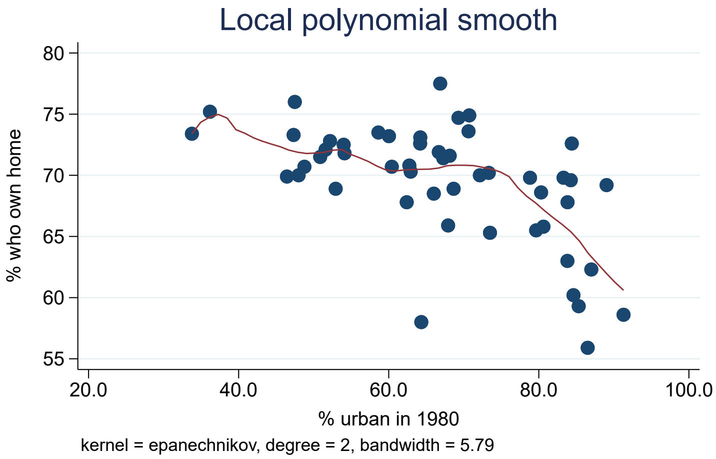
```
twoway (scatter ownhome pcturban80) (mband ownhome
pcturban80) (lpoly ownhome pcturban80)
```

Stata supports two other fit methods: median band (`mband`) and local polynomial smooth (`lpoly`). *Uses allstatesdc.dta*



```
lpoly ownhome pcturban80, degree(2)
```

We can also use the `lpoly` command on its own. Here we add the `degree(2)` option to specify a second-degree polynomial for the smoothing. See `help lpoly` for more options. *Uses allstatesdc.dta*

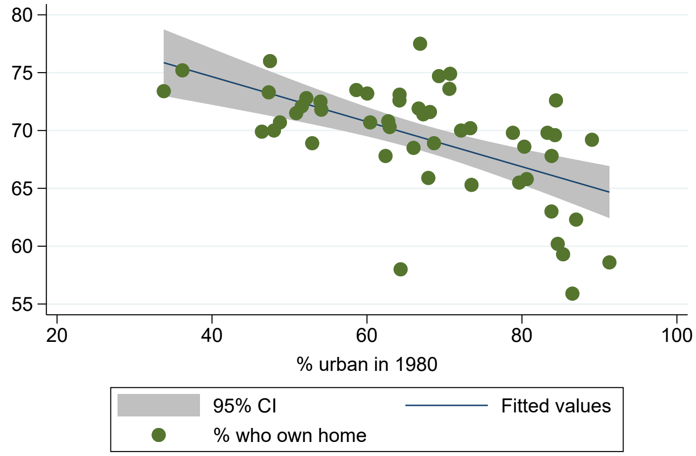

Local polynomial smooth
80
75
70
65
60
55
% who own home
20.0
40.0
60.0
80.0
100.0
% urban in 1980
kernel = epanechnikov, degree = 2, bandwidth = 5.79

2.3 Regression confidence interval fits

This section focuses on the twoway commands that are used for displaying confidence intervals (CI) around fit values: `lfitci`, `qfitci`, and `fpfitci`. The options permitted by these three commands are virtually identical, so I have selected `lfitci` to illustrate these options. (However, `fpfitci` does not permit the options `stdp`, `stdf`, and `stdr`.) For more information, see `help graph twoway lfitci`, `help graph twoway qfitci`, and `help graph twoway fpfitci`.

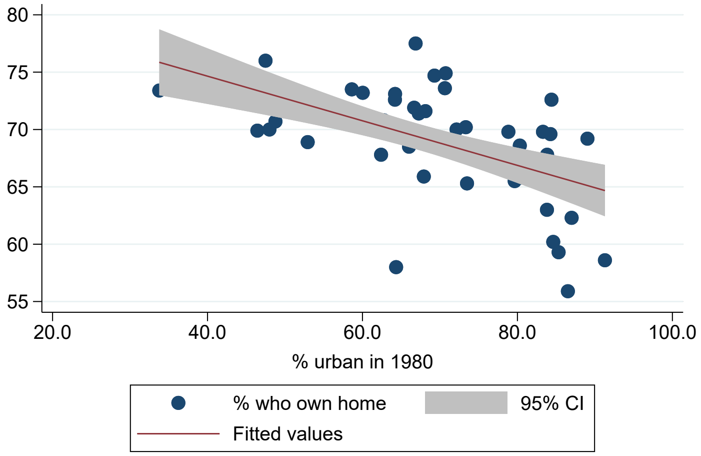
```
twoway (lfitci ownhome pcturban80) (scatter ownhome
pcturban80)
```

This graph uses the `lfitci` command to produce a linear fit with CI. The CI, by default, is computed using the standard error of prediction. We overlay this fit with a scatterplot. *Uses allstatesdc.dta*

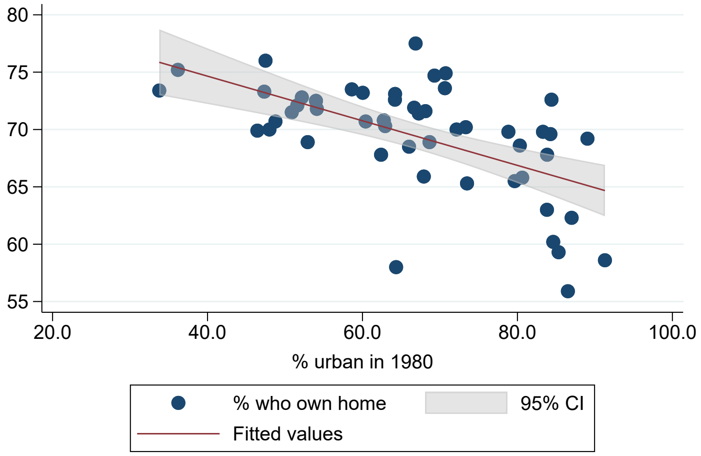


```
twoway (scatter ownhome pcturban80) (lfitci ownhome
pcturban80)
```

This example is the same as the previous example; however, the order of the `scatter` and `lfitci` commands is reversed. The order matters because the points that fell within the CI are not displayed, because they are masked by the shading of the CI. *Uses allstatesdc.dta*



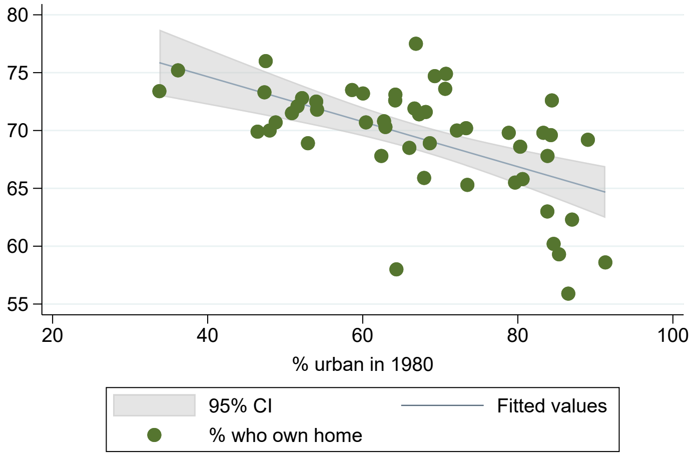
```
twoway (scatter ownhome pcturban80) (lfitci ownhome
pcturban80, bcolor(%40))
```



I do not recommend this as a solution to the previous problem but use it to illustrate the role of opacity when displaying confidence regions. I added the `bcolor(%40)` option to the `lfitci` command, which reduced the opacity of the confidence region to 30%. Note how we can see the markers through the confidence region. The problem is reduced but not eliminated. The markers should be displayed on top of the confidence region *Uses allstatesdc.dta*

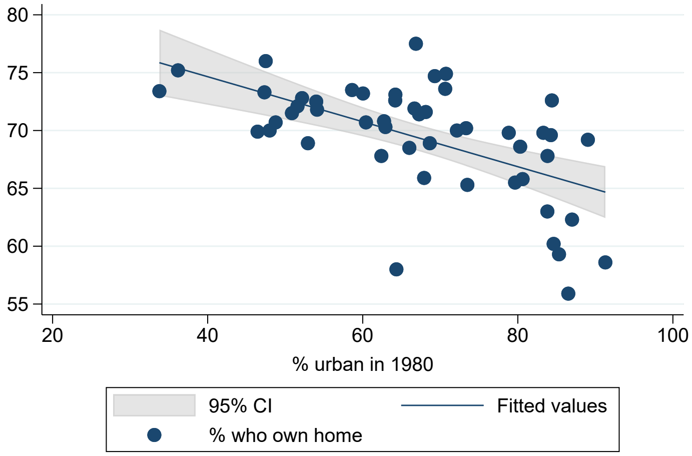
```
twoway (lfitci ownhome pcturban80, color(%40)) (scatter
ownhome pcturban80)
```

In this command, the `lfitci` command is issued first followed by the `scatter` command. Thus, the markers are displayed on top of the confidence region. But I preferred the navy-blue color from the prior example, when the `scatter` command was first. Instead, in this example, the scatterplot used the second *pstyle*, which shows markers in green. The next example shows a trick for handling this. For more information, see `help pstyle`. *Uses allstatesdc.dta*



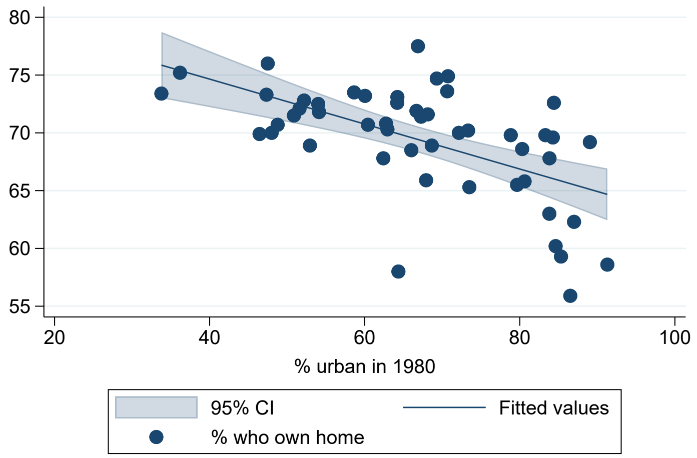
```
twoway (lfitci ownhome pcturban80, bcolor(%40)) (scatter
ownhome pcturban80), pcycle(1)
```

I have added the `pcycle(1)` option, which means that the *pstyle* returns back to 1 every time it advances. In the last example, the `scatter` results were shown with `pcycle(2)`, but now instead of advancing to 2, the `pcycle(1)` means the cycle resets after advancing past 1. *Uses allstatesdc.dta*

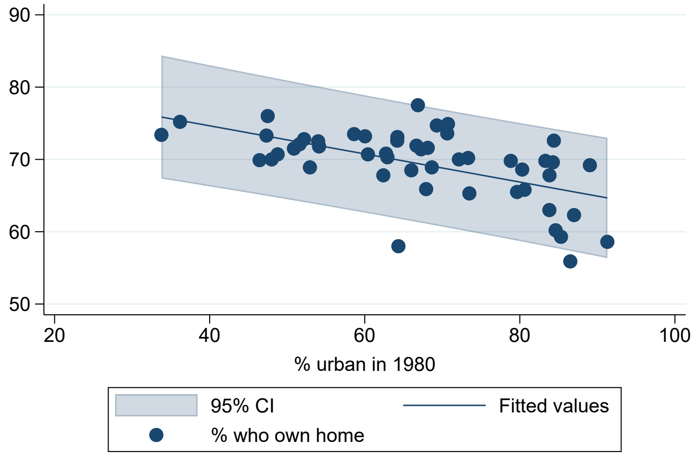


```
twoway (lfitci ownhome pcturban80, bcolor(navy%20)) (scatter
ownhome pcturban80), pcycle(1)
```

Forgive me for being finicky, but I really like to have the shaded color of the confidence region match the color of the markers and the fit line. By adding the `bcolor(navy%20)` option to the `lfitci` command, I show the confidence region in `navy` with 20% opacity. Note that I originally used 40% opacity but did not like how dark it was, so I switched to 20% opacity. *Uses allstatesdc.dta*



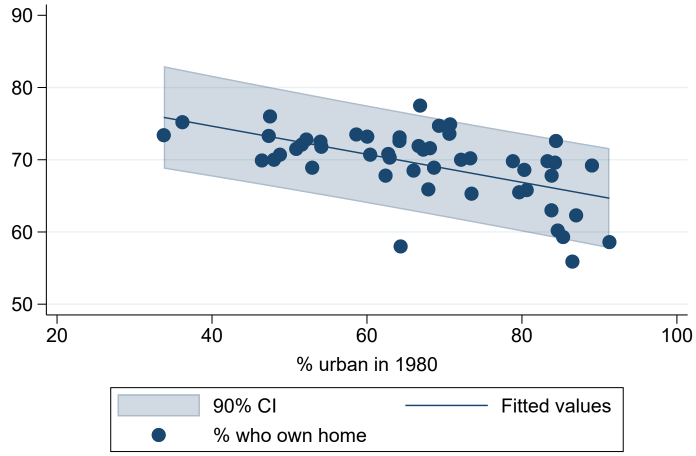
```
twoway (lfitci ownhome pcturban80, bcolor(navy%20) stdf)
(scatter ownhome pcturban80), pcycle(1)
```



Here we add the `stdf` option, which computes the CIs using the standard error of forecast. If samples were drawn repeatedly, this CI would capture 95% of the observations. With 50 observations, we would expect 2 or 3 observations to fall outside the CI. This expectation corresponds to the data shown here. *Uses allstatesdc.dta*

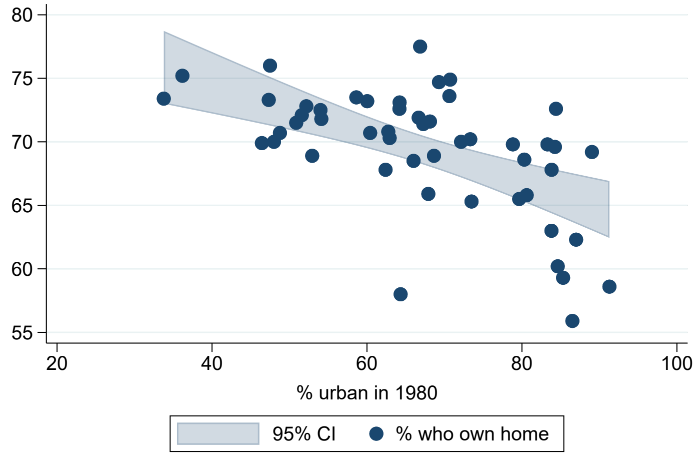
```
twoway (lfitci ownhome pcturban80, bcolor(navy%20) stdf
level(90)) (scatter ownhome pcturban80), pcycle(1)
```

We can use the `level()` option to set the confidence level for the CI. Here we make the confidence level 90%. The following examples will remove the `stdf level(90)` options so we can concentrate on other options. *Uses allstatesdc.dta*



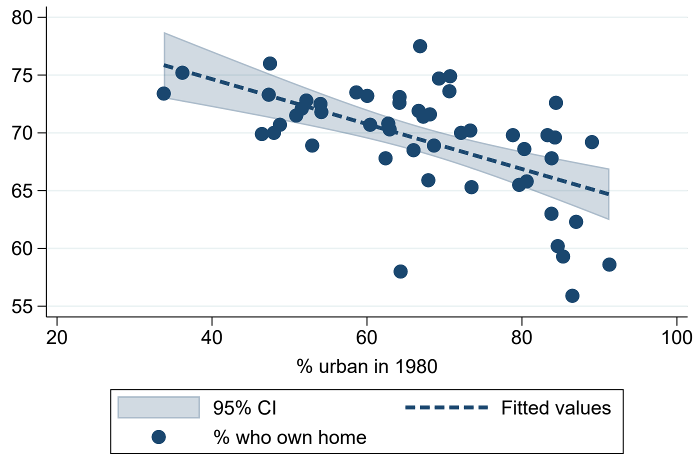
```
twoway (lfitci ownhome pcturban80, bcolor(navy%20) nofit)
(scatter ownhome pcturban80), pcycle(1)
```

We now look at how to control the display of the fit line. We can use the `nofit` option to suppress the display of the fit line. *Uses allstatesdc.dta*

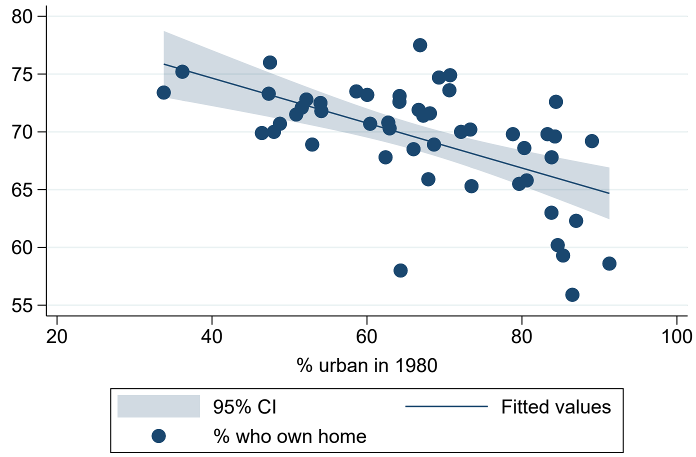


```
twoway (lfitci ownhome pcturban80, bcolor(navy%20)
clwidth(thick) clpattern(dash)) (scatter ownhome pcturban80),
pcycle(1)
```

We can supply options like `clwidth()` (line width), `clcolor()` (line color), and `clpattern()` (line pattern). This example shows a thick-dashed fit line. See **Styles : Linewidth** (section 10.7), **Styles : Colors** (section 10.2), and **Styles : Linepatterns** (section 10.6) for more details. *Uses allstatesdc.dta*



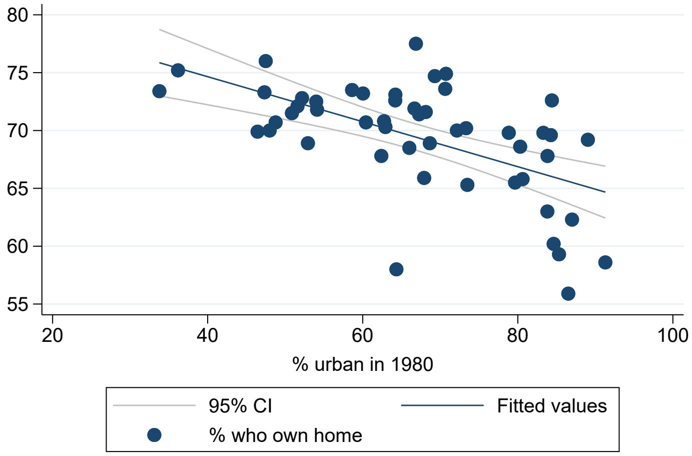
```
twoway (lfitci ownhome pcturban80, bcolor(navy%20)
alwidth(none)) (scatter ownhome pcturban80), pcycle(1)
```



You can customize the outline color of the CI with the `alcolor()`, `alwidth()`, and `alpattern()` options. In this example, I use `bcolor(navy%20)` to make the CI area `navy` with 40% opacity. Further, I add the `alwidth(none)` option to display the thickness of the outline as none (no line). See Twoway : Range (section 2.7) and `help graph twoway rarea` for more details. *Uses allstatesdc.dta*

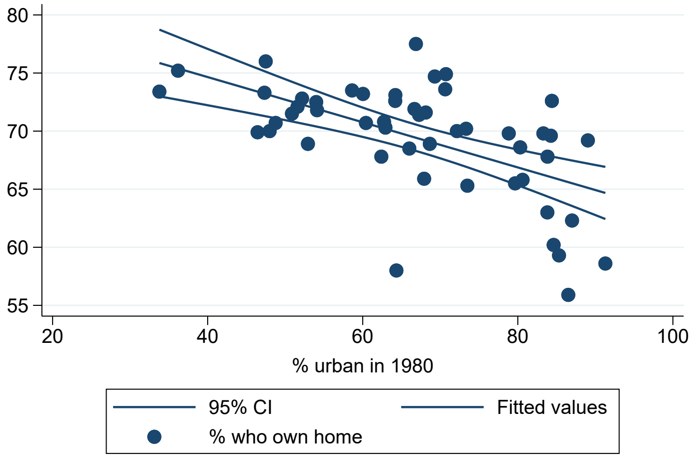
```
twoway (lfitci ownhome pcturban80, ciplot(rline)) (scatter
ownhome pcturban80), pcycle(1)
```

The `ciplot()` option selects a different command for displaying the CI; the default is `ciplot(rarea)`. Here we use the `ciplot(rline)` option to display the CI as two lines without a filled area. The valid options include `rarea`, `rbar`, `rspike`, `rcap`, `rcapsym`, `rscatter`, `rline`, and `rconnected`. *Uses allstatesdc.dta*



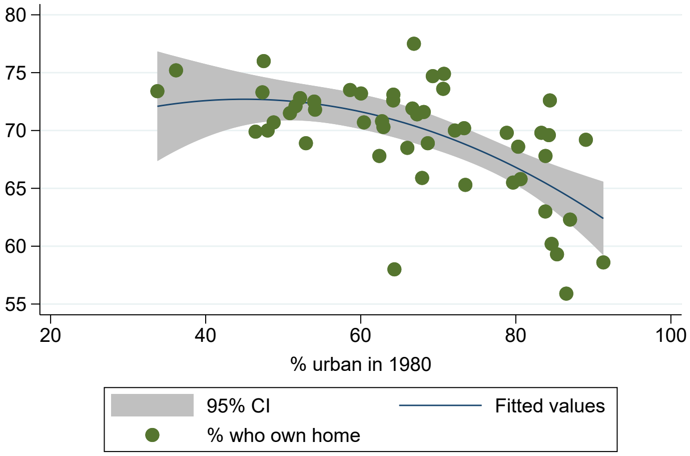
```
twoway (lfitci ownhome pcturban80, ciplot(rline) lcolor(navy)
lwidth(medthick)) (scatter ownhome pcturban80), pcycle(1)
```

This example adds the `lcolor(navy)` and `lwidth(medthick)` options to make both the fit and CI lines medium thick and `navy`. *Uses allstatesdc.dta*



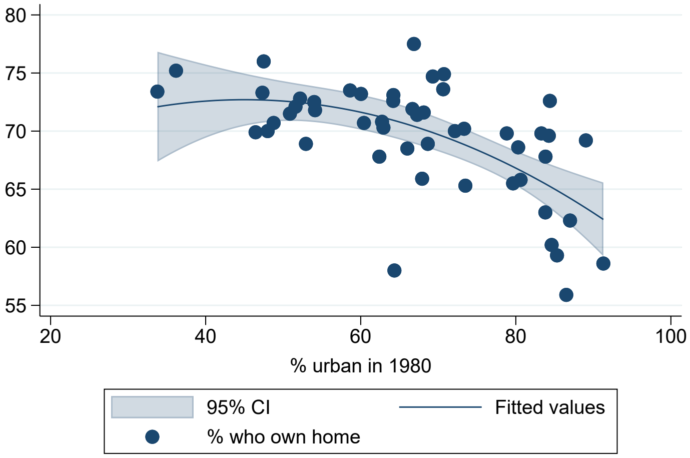
```
twoway (qfitci ownhome pcturban80) (scatter ownhome
pcturban80)
```

In addition to the `lfitci` command, the `qfitci` command produces a quadratic fit with a CI. We overlay this fit with a scatterplot. *Uses allstatesdc.dta*



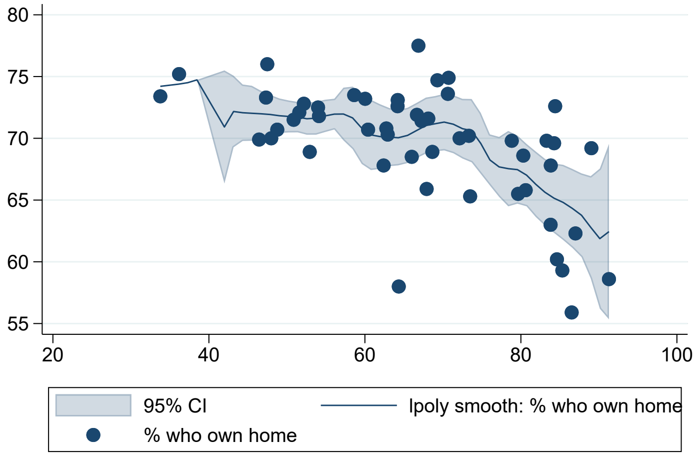
```
twoway (qfitci ownhome pcturban80, bcolor(navy%20)) (scatter
ownhome pcturban80), pcycle(1)
```

I prefer the look using the `bcolor(navy%20)` option for shading the CI and the `pcycle(1)` option to display the fit line and markers both using `navy`. *Uses allstatesdc.dta*



```
twoway (lpolyci ownhome pcturban80, bcolor(navy%20)) (scatter
ownhome pcturban80), pcycle(1)
```

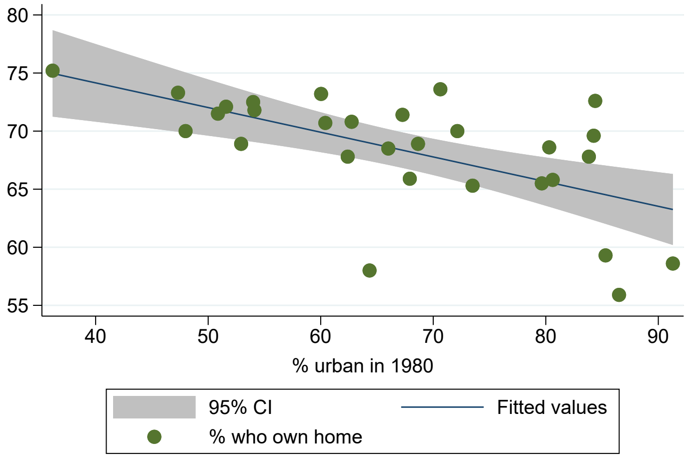
We can also use the `lpolyci` command to produce a local polynomial smooth with a CI. *Uses allstatesdc.dta*



Say we wanted to produce scatterplots with linear fits and confidence intervals for northern states and nonnorthern states. The goal would be to produce an overlapping graph where the markers and confidence regions for nonnorthern states are shown in blue and northern states are shown in red. As a first step, let's create a scatterplot and linear fit (with confidence regions) for nonnorthern states.

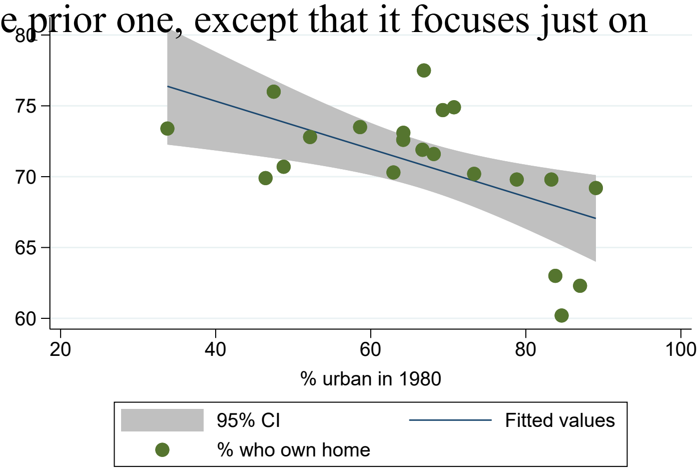
```
twoway (lfitci ownhome pcturban80 if north==0) (scatter
ownhome pcturban80 if north==0)
```

This command shows the scatterplot, linear fit, and confidence region predicting home ownership from the percent urban, focusing just on nonnorthern states. *Uses allstatesdc.dta*

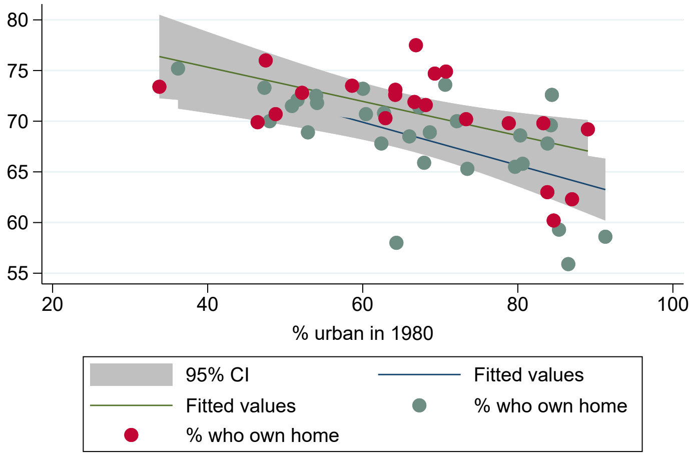


```
twoway (lfitci ownhome pcturban80 if north==1) (scatter
ownhome pcturban80 if north==1)
```

This command is exactly like the prior one, except that it focuses just on northern states. *Uses allstatesdc.dta*



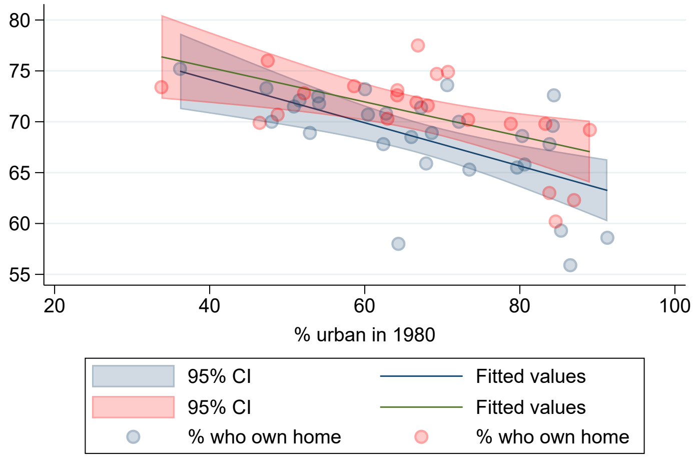
```
twoway (lfitci ownhome pcturban80 if north==0) (lfitci
ownhome pcturban80 if north==1) (scatter ownhome pcturban80
if north==0) (scatter ownhome pcturban80 if north==1)
```



This command overlays each of the two prior plots into a single graph. It shows the linear fit and CI for the nonnorthern states, followed by the linear fit and CI for the northern states. It then shows the scatterplot for the nonnorthern states and then the scatterplot for the northern states. But the confidence region for the northern states conceals the confidence region for the nonnorthern states. *Uses allstatesdc.dta*

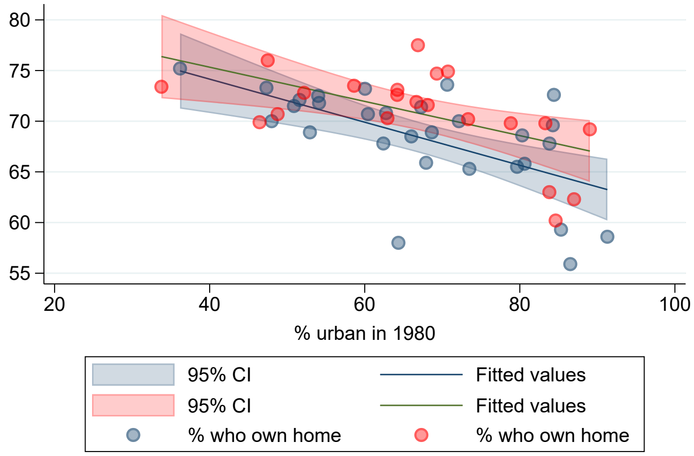
```
twoway (lfitci ownhome pcturban80 if north==0,
bcolor(navy%20)) (lfitci ownhome pcturban80 if north==1,
bcolor(red%20)) (scatter ownhome pcturban80 if north==0,
mcolor(navy%20)) (scatter ownhome pcturban80 if north==1,
mcolor(red%20))
```

I have repeated the command from above but have used `navy%20` as the color for the markers and confidence regions for the nonnorthern states and `red%20` for the markers and confidence regions for the northern states. I think the markers are too transparent. The next example increases the opacity for the markers.
Uses allstatesdc.dta



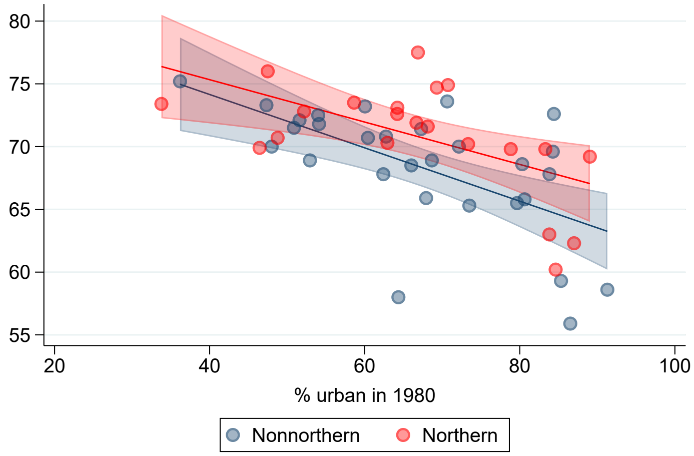
```
twoway (lfitci ownhome pcturban80 if north==0,
bcolor(navy%20)) (lfitci ownhome pcturban80 if north==1,
bcolor(red%20)) (scatter ownhome pcturban80 if north==0,
mcolor(navy%40)) (scatter ownhome pcturban80 if north==1,
mcolor(red%40))
```

I have repeated the command from above but have used 40% opacity for the markers. That is, `mcolor(navy%40)` for the nonnorthern states and `mcolor(red%40)` for the northern states. Now we can clearly see both the markers and confidence regions for each group.
Uses allstatesdc.dta



```
tw (lfitci ownhome pcturban80 if north==0, lcolor(navy)
bcolor(navy%20)) (lfitci ownhome pcturban80 if north==1,
lcolor(red) bcolor(red%20)) (scatter ownhome pcturban80 if
north==0, mcolor(navy%40)) (scatter ownhome pcturban80 if
north==1, mcolor(red%40)), legend(order(5 "Nonnorthern" 6
"Northern"))
```

In this command, I made two cosmetic fixes. First, I specified the color of the fit line as `lcolor(navy)` for the nonnorthern states and `lcolor(red)` for the northern states. Second, I added the `legend()` option to display only the 5th and 6th elements of the legend, labeling them "Nonnorthern" and "Northern" (respectively). *Uses allstatesdc.dta*

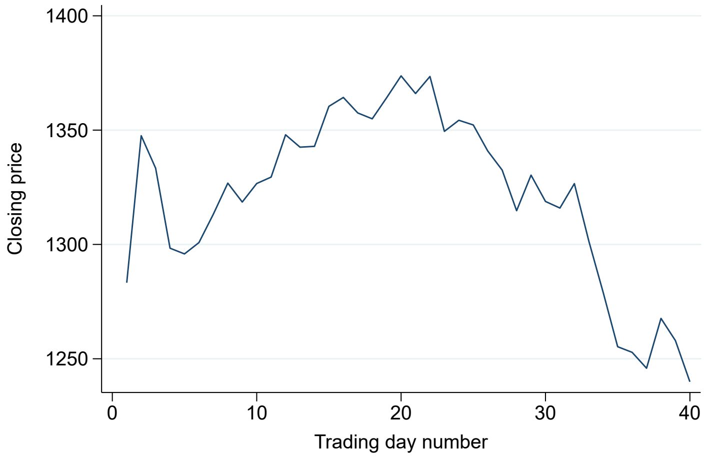


2.4 Line plots

This section focuses on the twoway commands for creating line plots, including the `twoway line` and `twoway connected` commands. The `line` command is the same as `scatter`, except that the points are connected by default and marker symbols are not permitted, whereas the `twoway connected` command permits marker symbols. This section also illustrates `twoway tsline` and `twoway tsrline`, which are useful for drawing line plots when the x variable is a date variable. Because all these commands are related to the `twoway scatter` command, they support most of the options you would use with `twoway scatter`. For more information, see `help graph twoway line`, `help graph twoway connected`, and `help graph twoway tsline`.

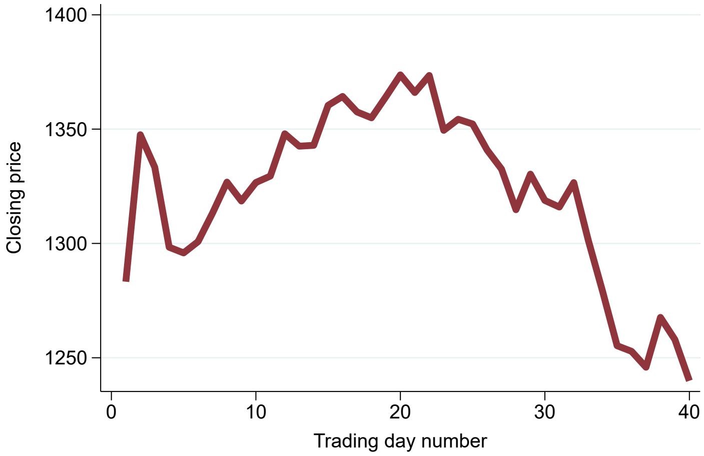
```
twoway line close tradeday, sort
```

In this example, we use `twoway line` to show the closing price across trading days. The inclusion of the `sort` option is recommended when we have points connected in a Stata graph. *Uses spjanfeb2001.dta*



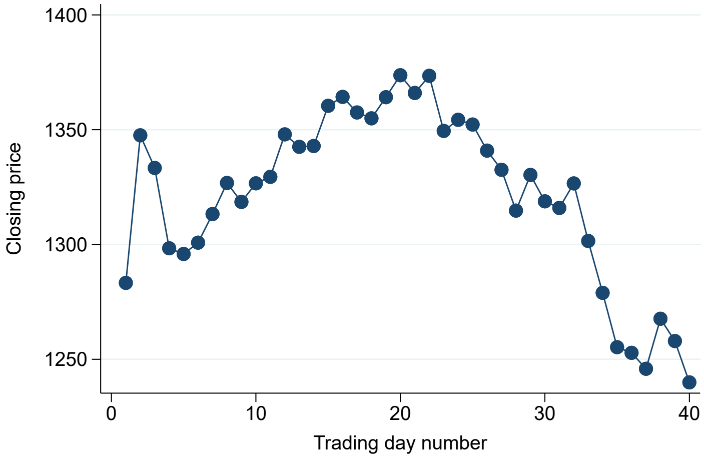
```
twoway line close tradeday, sort lwidth(vthick)
lcolor(maroon)
```

Here we show options controlling the width and color of the lines. By using `lwidth(vthick)` (line width) and `lcolor(maroon)` (line color), we make the line very thick and maroon. For more examples on selecting line widths, see Styles : Linewidth (section 10.7), and see See Styles : Colors (section 10.2) for more details about colors (including intensity and opacity). *Uses spjanfeb2001.dta*



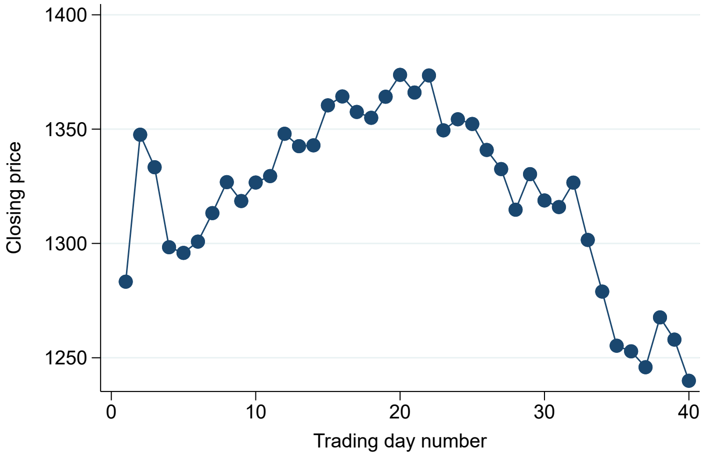
```
twoway connected close tradeday, sort
```

This `twoway connected` graph is similar to the `twoway line` graphs above, except that for `connected`, a marker is shown for each data point. *Uses spjanfeb2001.dta*



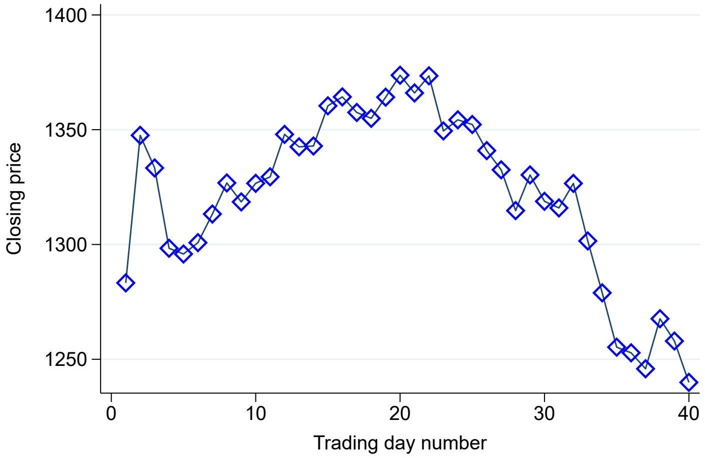
```
twoway scatter close tradeday, connect(l) sort
```

This graph is identical to the previous one, except this graph is made with the `scatter` command using the `connect(l)` option. This illustrates the convenience of using `connected` because we do not need to manually specify the `connect()` option. *Uses spjanfeb2001.dta*



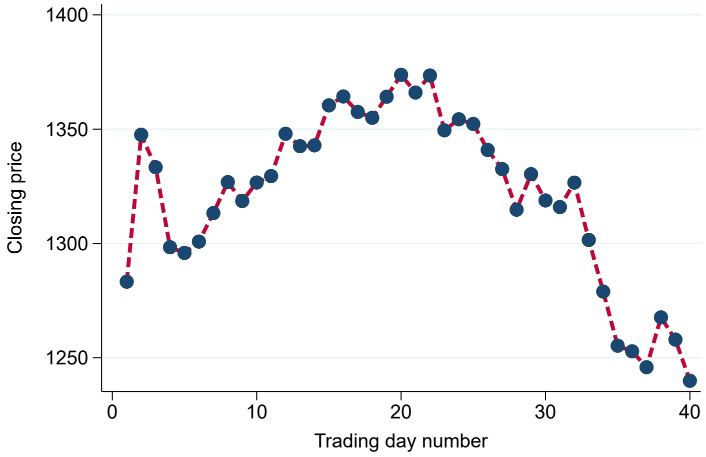
```
twoway connected close tradeday, sort msymbol(Dh)
mcolor(blue) msize(large)
```

To control the marker symbols, we can use the options `msymbol()`, `mcolor()`, and `msize()`. Here we make the symbols large, blue, hollow diamonds. See **Options : Markers** (section 8.1) for more examples. *Uses spjanfeb2001.dta*



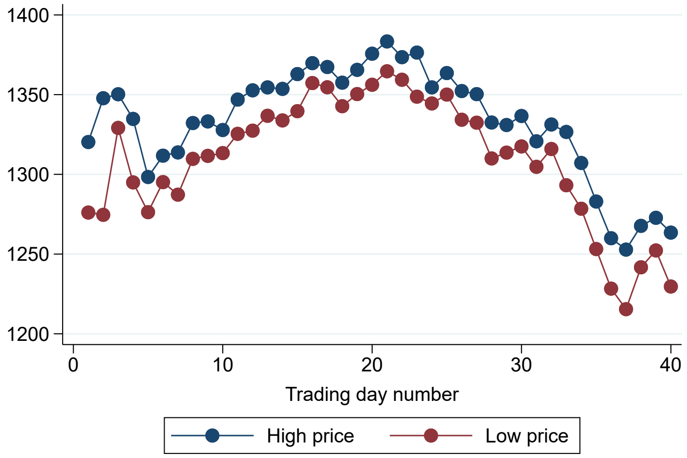
```
twoway connected close tradeday, sort lcolor(cranberry)
lpattern(dash) lwidth(thick)
```

We can control the look of the lines with connect options such as `lcolor()`, `lpattern()` (line pattern), and `lwidth()`. Here we make the line cranberry, dashed, and thick. See **Options : Connecting** (section 8.3) for more details on connecting points. *Uses spjanfeb2001.dta*



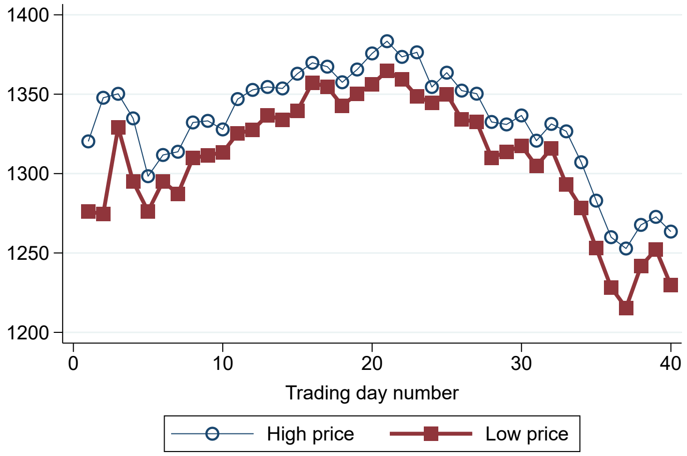
```
twoway connected high low tradeday, sort
```

We can graph multiple variables on one graph. Here we graph the high and low prices across trading days. *Uses spjanfeb2001.dta*



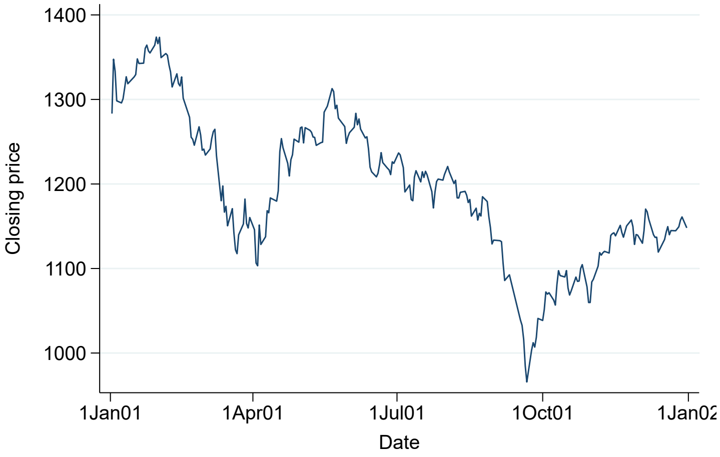
```
twoway connected high low tradeday, sort lwidth(thin thick)
msymbol(Oh S)
```

When graphing multiple variables, we can specify connect and marker symbol options to control each line. In this example, we use a thin line for the high price and a thick line for the low price. We also differentiate the two lines by using different marker symbols: hollow circles for the high price and squares for the low price. *Uses spjanfeb2001.dta*



Stata has more commands for creating line plots where the x variable is a date variable, namely, `twoway tsline` and `twoway tsrline`. The `tsline` command is similar to the `line` command, and the `tsrline` is similar to the `rline` command. However, both of these `ts` commands offer extra features, making it easier to reference the x variable for dates (see `help graph twoway tsline`). To illustrate these commands, let's use the `sp2001ts` data file, which has the prices for the S&P 500 index for 2001 with the trading date stored as a date variable named `date`. Before saving the file `sp2001ts`, the `tsset date, daily` command was used to tell Stata that the variable `date` represents the time variable and that it represents daily data.

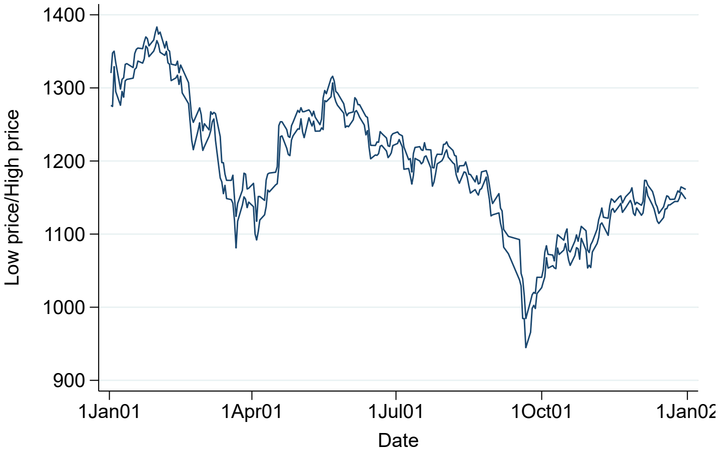
```
twoway tsline close
```



The `tsline` (time-series line) graph shows the closing price on the y axis and the date on the x axis. We did not specify the x variable in the graph command. Stata knew the variable representing time because we previously issued the `tsset date, daily` command before saving the `sp2001ts` file. If we save the data file, Stata remembers the time variable, and we do not need to set it again. *Uses sp2001ts.dta*

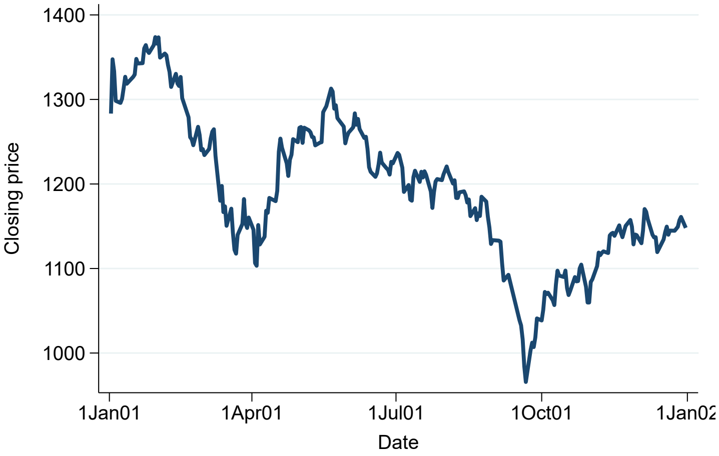
```
twoway tsrline low high
```

We can also use the `tsrline` (time-series range) graph to show the low price and high price for each day. *Uses sp2001ts.dta*



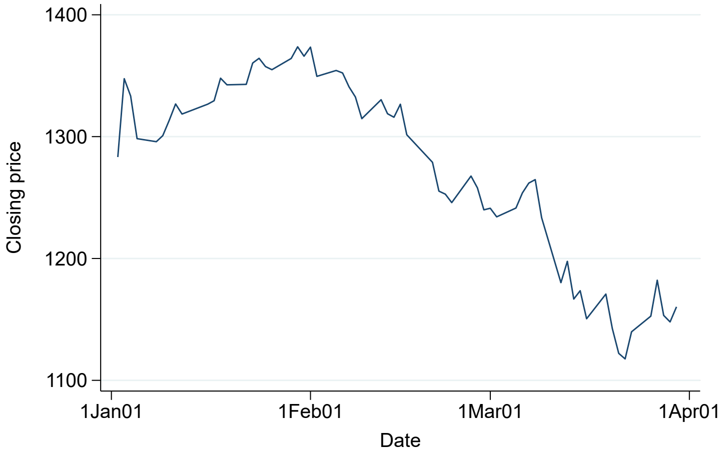
```
twoway tsline close, lwidth(thick) lcolor(navy)
```

As with `twoway line`, we can use connect options to control the line. Here we make the line thick and navy. See **Styles : Colors** (section 10.2) for more details about colors (including intensity and opacity). Also see **Styles : Linewidth** (section 10.7) for details in specifying line widths (including the ability to specify widths in points). *Uses sp2001ts.dta*



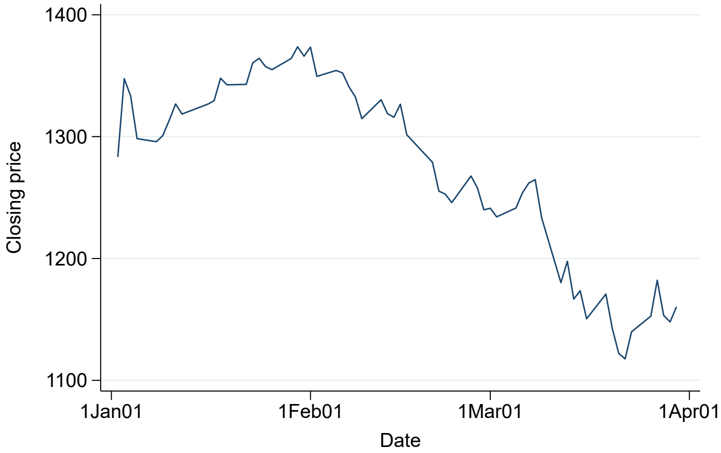
```
twoway tsline close if (date >= mdy(1,1,2001)) & (date <=
mdy(3,31,2001))
```

We can use `if` to subset cases to graph. Here we graph the closing prices between January 1, 2001, and March 31, 2001. See the next example for an easier way of doing this. *Uses sp2001ts.dta*



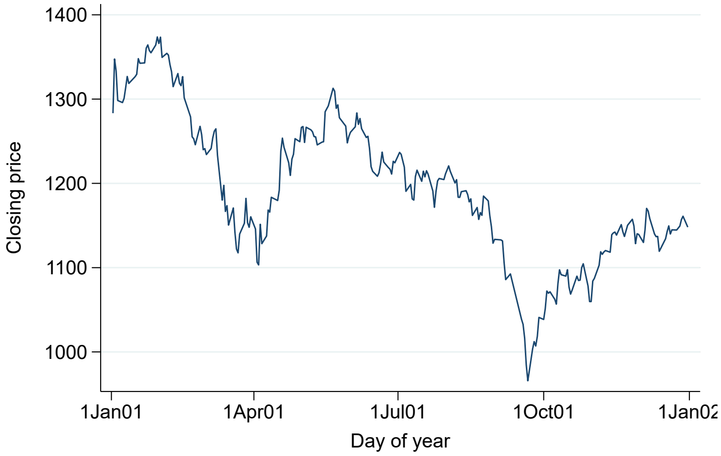
```
twoway tsline close if tin(01jan2001,31mar2001)
```

When using the `tsline` command, we can use `tin()` (time in between) to specify that we want to graph just the cases between January 1, 2001, and March 31, 2001, inclusively. *Uses sp2001ts.dta*



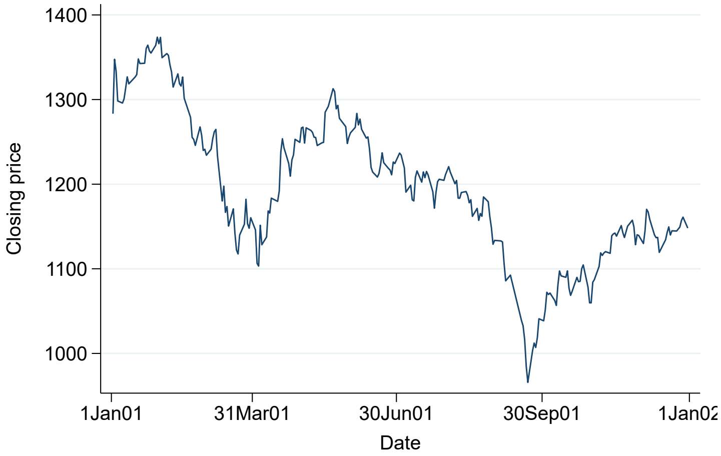
```
twoway tsline close, ttitle(Day of year)
```

The `ttitle()` (time title) option gives a title to the time variable. We specify this as a `ttitle()` instead of `xtitle()` because this refers to the axis with the time variable. *Uses sp2001ts.dta*



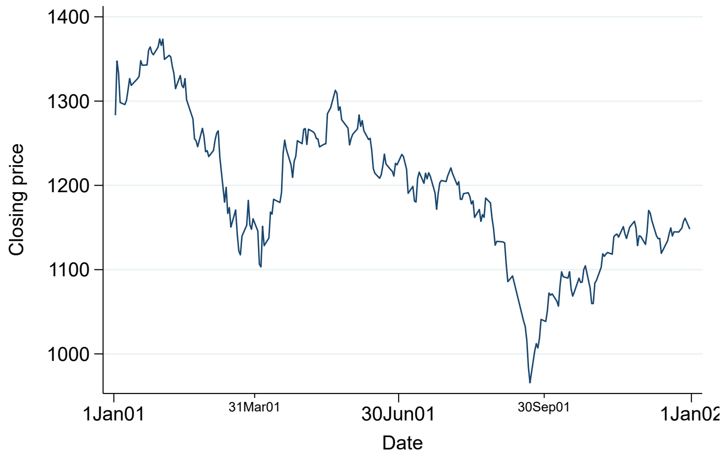
```
twoway tsline close, tlabel(01jan2001 31mar2001 30jun2001
30sep2001 01jan2002)
```

The `tlabel()` option labels the time points on the time axis. Here we specify these dates by using date literals; Stata knows how to interpret and appropriately label the graph with these values. *Uses sp2001ts.dta*



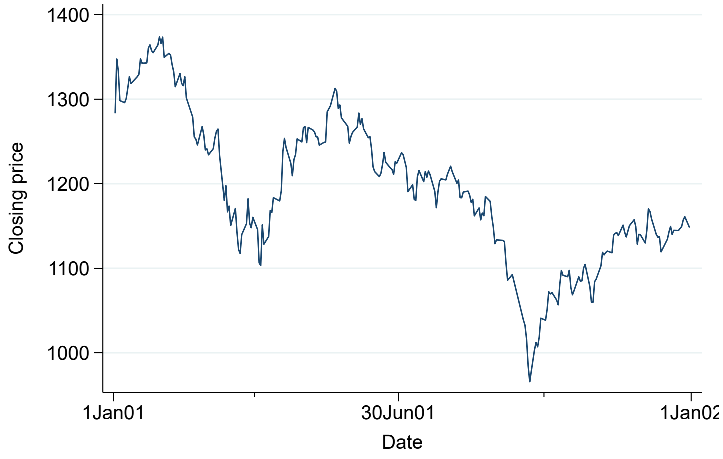
```
twoway tsline close, tlabel(01jan2001 30jun2001 01jan2002)
tmlabel(31mar2001 30sep2001)
```

In this example, we use the `tmlabel()` option to include minor labels. *Uses sp2001ts.dta*



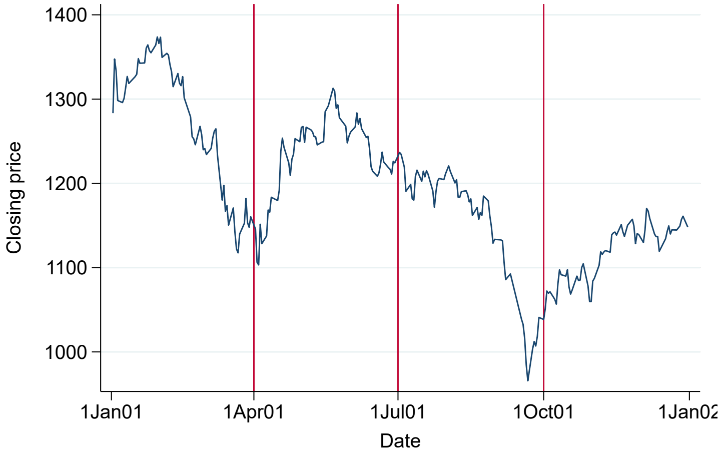
```
twoway tsline close, tlabel(01jan2001 30jun2001 01jan2002)
tmtick(31mar2001 30sep2001)
```

Instead of minor labels, we now use the `tmtick()` option to include minor ticks. *Uses sp2001ts.dta*



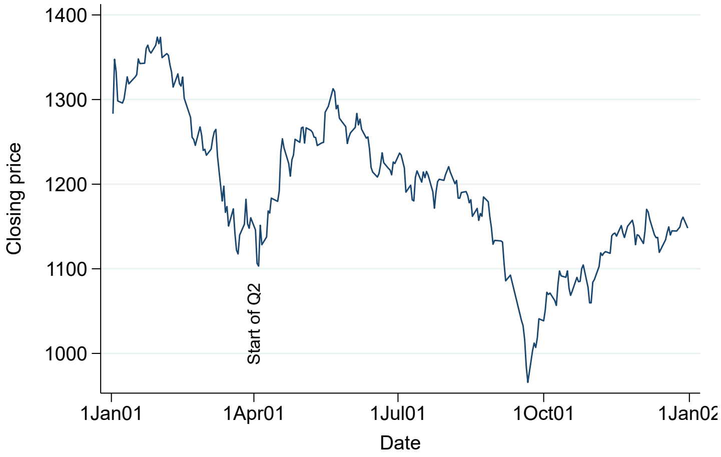
```
twoway tsline close, tline(01apr2001 01jul2001 01oct2001)
```

We can use the `tline()` option to include lines at certain time points. Here we place lines at the start of the second, third, and fourth quarters. *Uses sp2001ts.dta*



```
twoway tsline close, ttext(1035 01apr2001 "Start of Q2",
orientation(vertical))
```

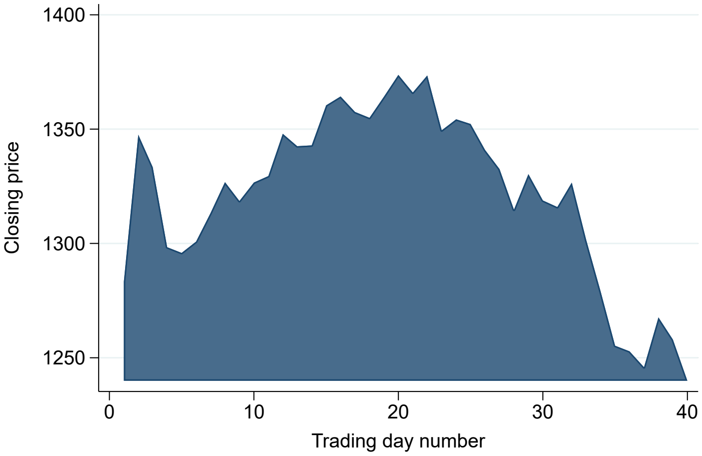
The `ttext()` option adds text to the graph. The first coordinate refers to the position on the y axis, and the second coordinate is the position on the time axis for the date. *Uses sp2001ts.dta*



2.5 Area plots

This section illustrates the use of area graphs using `twoway area`. These graphs are similar to `twoway line` graphs, except that the area under the line is shaded. As a result, many of the options that you would use with `twoway line` are applicable; see Twoway : Line (section 2.4) for more details. For even more details, see `help graph twoway area`. These examples use the `spjanfeb2001` data file, which has the prices for the S&P 500 index for January and February 2001.

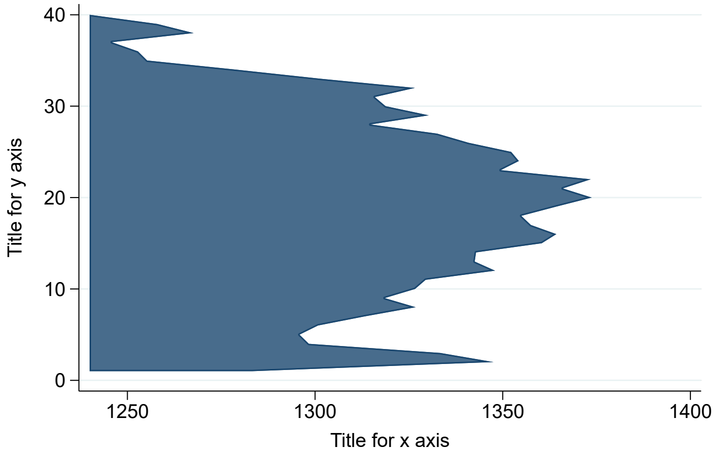
```
twoway area close tradeday, sort
```



This is an example of a `twoway area` graph. Because this graph is composed of connected points, the `sort` option is recommended in case the data are not already sorted by `tradeday`. If the data are not sorted and the `sort` option is not specified, the points are connected in the order they appear in the data file producing a graph that you do not desire. *Uses spjanfeb2001.dta*

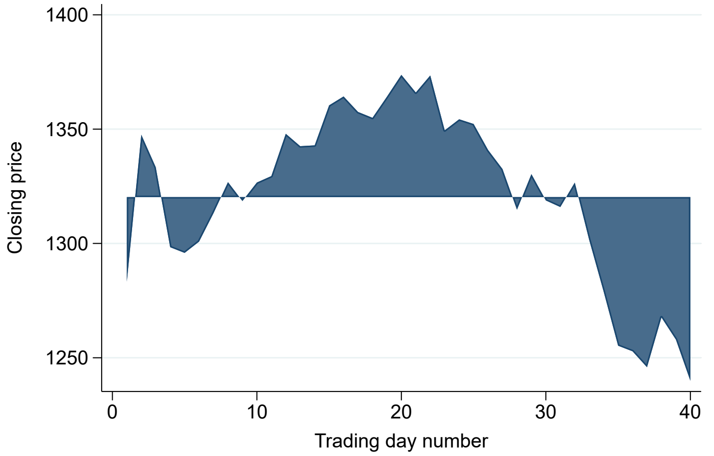
```
twoway area close tradeday, horizontal sort xtitle(Title for
x axis) ytitle(Title for y axis)
```

The `horizontal` option swaps the position of the `close` and `tradeday` variables. The x axis remains at the bottom, and the y axis remains at the left. *Uses spjanfeb2001.dta*



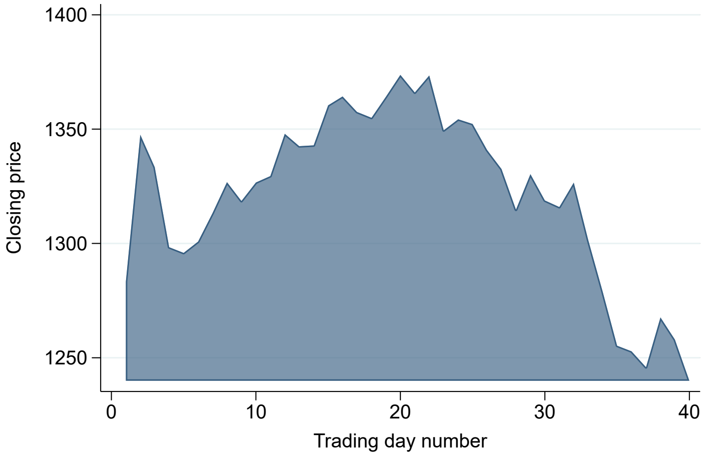
```
twoway area close tradeday, sort base(1320.28)
```

The `base()` option indicates a base from which the area is to be shaded. Here the base is the closing price on the first trading day, and thus all the subsequent points are a kind of deviation from the first day's closing price. *Uses spjanfeb2001.dta*



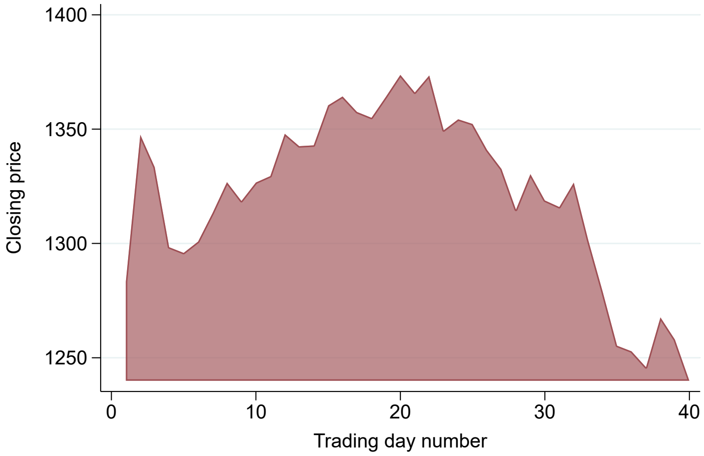
```
twoway area close tradeday, sort bcolor(navy%70)
```

The `bcolor()` option sets the color of the shaded area and the line. Here we color the shaded area and the line using `navy` with 70% opacity. *Uses spjanfeb2001.dta*



```
twoway area close tradeday, sort bcolor(maroon%70)
```

The `bcolor()` option sets the color of the shaded area and the line. Here we color the shaded area and the line `maroon` with 70% opacity. Although they are not shown, you can also use the `fcolor()` and `lcolor()` options to control the fill color and line color and the `lwidth()` option to control the thickness of the outline. *Uses spjanfeb2001.dta*

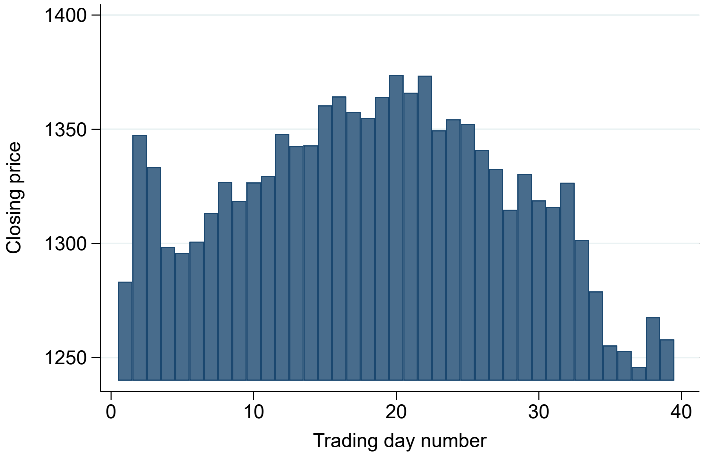


2.6 Bar plots

This section illustrates the use of twoway bar graphs using `twoway bar`. These graphs show a bar for each x value, where the height of the bar corresponds to the value of the y variable. For more details, see `help graph twoway bar`. We will continue to use the `spjanfeb2001` data file, which has the prices for the S&P 500 index for January and February 2001. The `twoway bar` command is useful for creating bar graphs with overlays of lines, points, or other plot types and can be used with evenly spaced x-variable data. `graph bar` is more useful for creating bar graphs with categorical data.

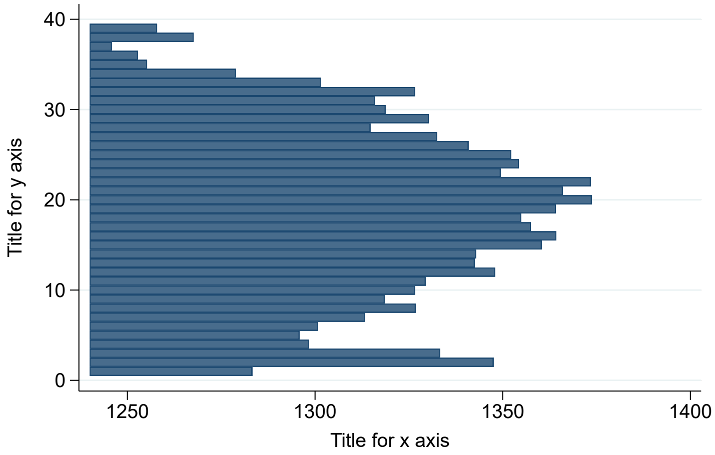
```
twoway bar close tradeday
```

Consider this `bar` chart, which shows the closing prices of the S&P 500 broken down by the trading day of the year. *Uses spjanfeb2001.dta*

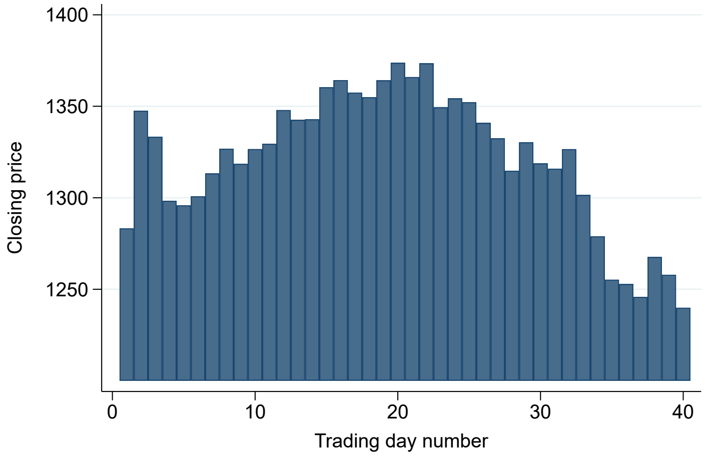


```
twoway bar close tradeday, horizontal xtitle(Title for x
axis) ytitle(Title for y axis)
```

We can make the `close` and `tradeday` variables trade places by specifying the `horizontal` option. The x axis remains at the bottom, and the y axis remains at the left. *Uses spjanfeb2001.dta*



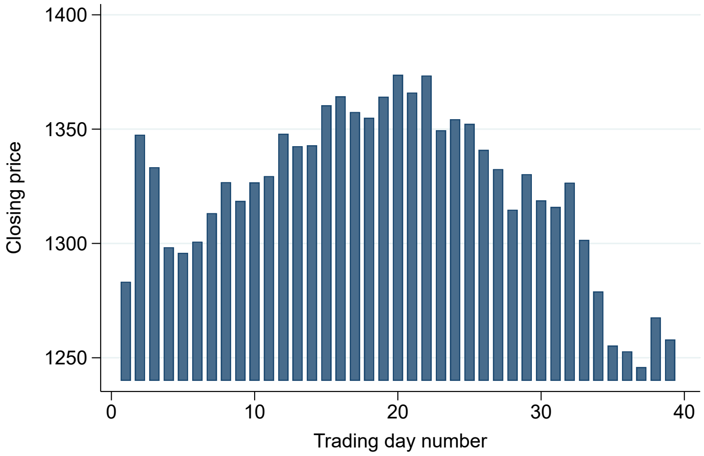
```
twoway bar close tradeday, base(1200)
```



The base for the bar charts is the trading day with the lowest price, unless otherwise specified. The closing price on day 40 was 1,239.94, so if you had not specified the `base()` option, the base would have been 1,239.94. As a result, the bar for day 40 would have had a zero height. However, because we specified `base(1200)`, the base height is 1,200. *Uses spjanfeb2001.dta*

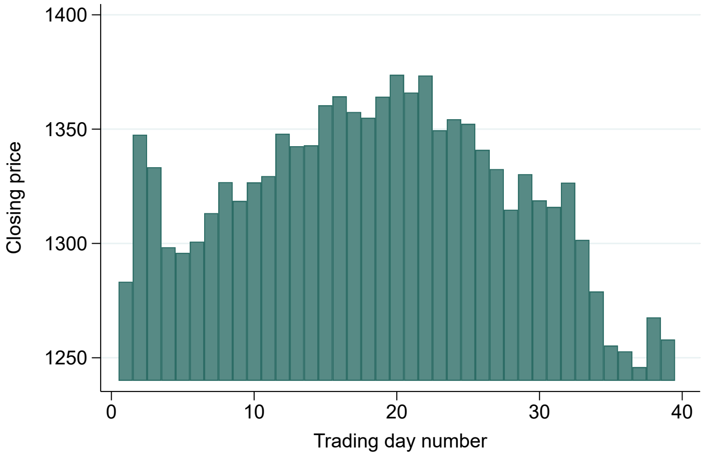
```
twoway bar close tradeday, barwidth(.7)
```

By default, the width of each bar is one x unit (here, one day). By making the width of the bars 0.7, we can obtain a small gap between the bars. *Uses spjanfeb2001.dta*



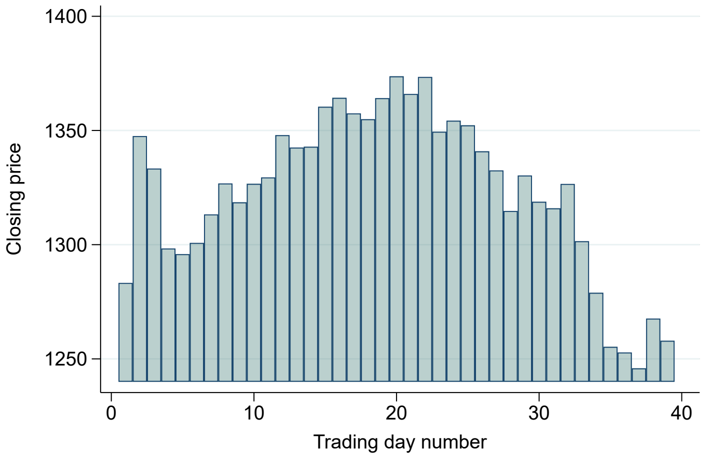
```
twoway bar close tradeday, bcolor(emerald)
```

In this example, the `bcolor(emerald)` option is used to make the bar color `emerald`. See **Styles : Colors** (section 10.2) for more details about colors (including intensity and opacity). *Uses spjanfeb2001.dta*



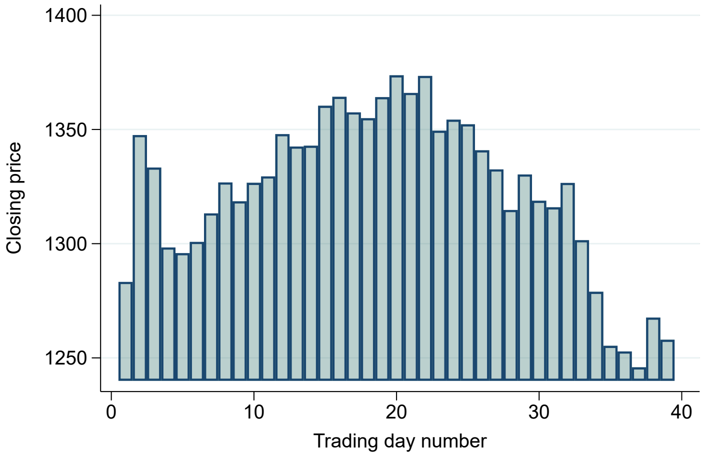
```
twoway bar close tradeday, fcolor(emerald%40) lcolor(navy)
```

This example uses the `fcolor()` (fill color) option to set the color of the inside of the bars, and the `lcolor()` (line color) option to set the color of the bar outlines. Here we make the fill color `emerald` with 40% opacity, and the outline color `navy`. *Uses spjanfeb2001.dta*



```
twoway bar close tradeday, fcolor(emerald%40) lcolor(navy)
lwidth(medthick)
```

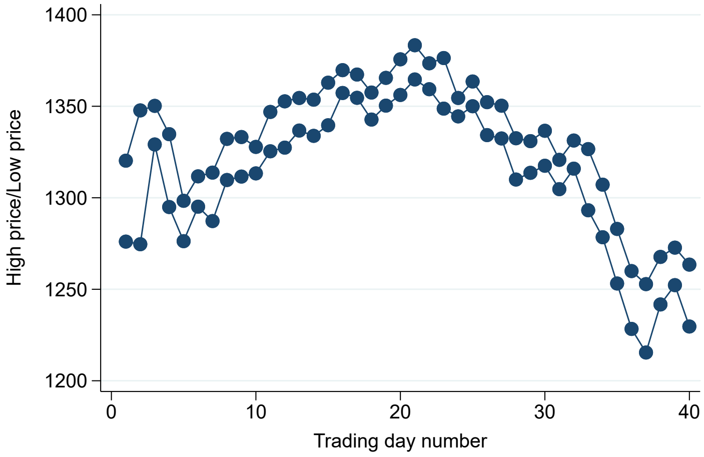
This example adds the `lwidth()` (line width) option to set the width of the bar outlines. Here we specify `lwidth(medthick)`, which yields bar outlines thicker than the prior example. *Uses spjanfeb2001.dta*



2.7 Range plots

This section focuses on twoway commands that display range plots. The major characteristic these graphs share is that, for each x value, there are two corresponding y values. A common example is a confidence interval where, for each x value, there are upper and lower confidence limits. We will first look at examples of all these types of graphs and then consider the options we can use to customize them. For more information, see `help graph twoway rarea`, `help graph twoway rbar`, `help graph twoway rspike`, `help graph twoway rcap`, `help graph twoway rcapsym`, `help graph twoway rscatter`, `help graph twoway rline`, and `help graph twoway rconnected`. Let's start by looking at the `rconnected`, `rscatter`, `rline`, and `rarea` graphs, which use combinations of lines, symbols, and shading to display range plots. These examples use the `spjanfeb2001` data file.

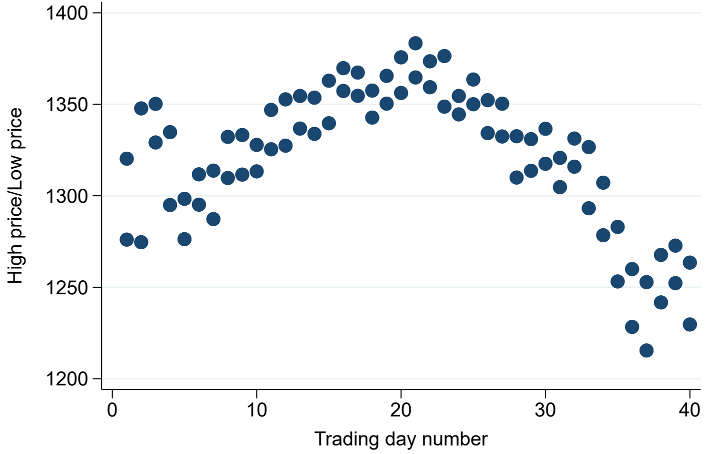
```
twoway rconnected high low tradeday, sort
```



The `rconnected` (range connected) graph shows the high and low prices by `tradeday`, the number of days stocks have been traded in the year. The `rconnected` plot shows a separate line for the high and low prices, and a marker appears for each x value. The `sort` option is recommended because the points are connected by lines and is needed if the data were not already sorted on `tradeday`. *Uses spjanfeb2001.dta*

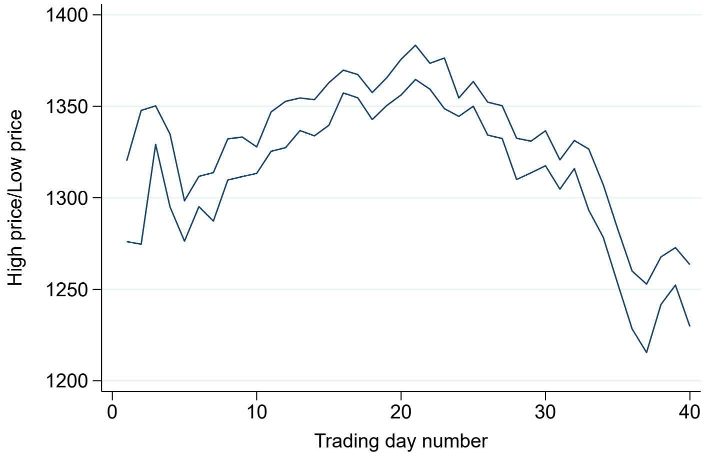
```
twoway rscatter high low tradeday
```

The `rscatter` graph is similar to the `rconnected` graph, except that lines connecting the symbols are not plotted. *Uses spjanfeb2001.dta*



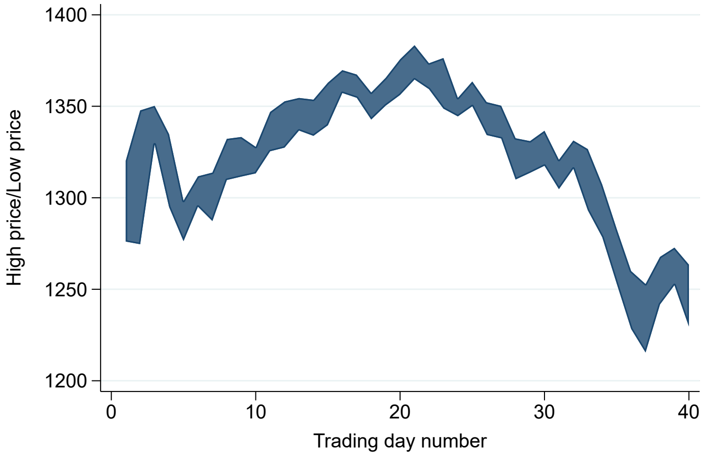
```
twoway rline high low tradeday, sort
```

The `rline` graph is similar to the `rconnected` graph, except that symbols are not plotted at each level of x. Note the inclusion of the `sort` option. This option is recommended because the points are connected by lines and is needed if the data were not already sorted on `tradeday`. *Uses spjanfeb2001.dta*



```
twoway rarea high low tradeday, sort
```

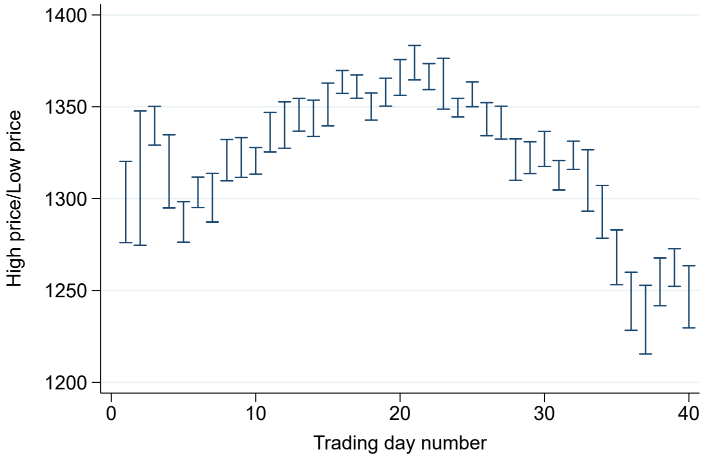
The `rarea` graph is similar to the `rline` graph, except that we can control the fill color of the area between the high and low values. *Uses spjanfeb2001.dta*



Now let's look at `rcap`, `rspike`, and `rcapsym` graphs, which use combinations of spikes, caps, and symbols to display range plots. Then we will look at `rbar` graphs, which use bars to display range plots.

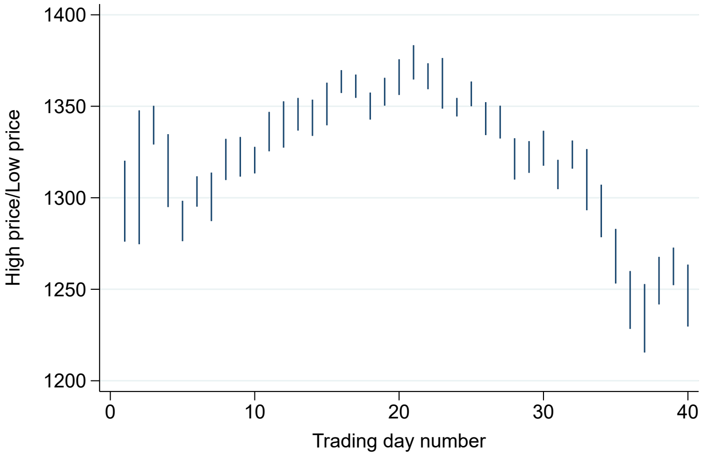
```
twoway rcap high low tradeday
```

The `rcap` graph shows a spike ranging from the low to high values and puts a cap at the top and bottom of each spike. *Uses spjanfeb2001.dta*



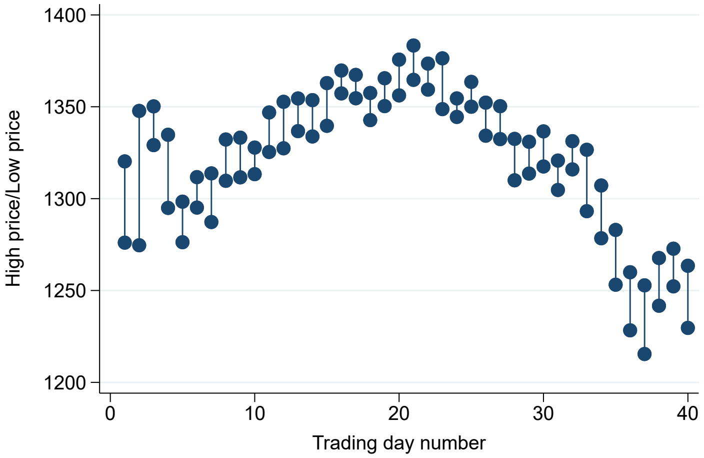
```
twoway rspike high low tradeday
```

The `rspike` graph is similar to the `rcap` graph, except that no caps are placed on the spikes. *Uses spjanfeb2001.dta*



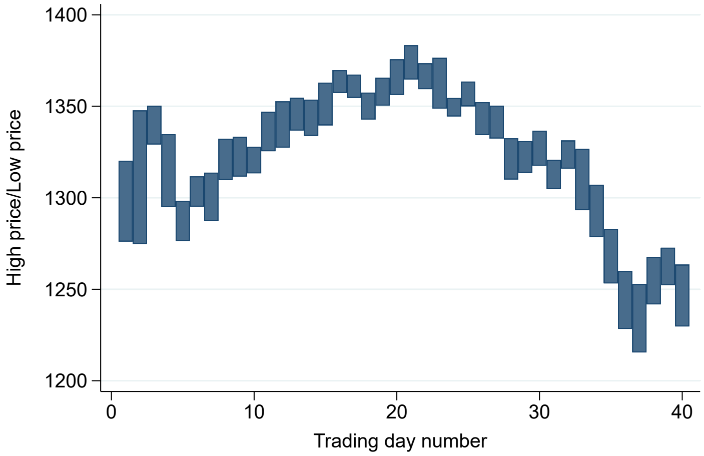
```
twoway rcapsym high low tradeday
```

The `rcapsym` graph is similar to the `rcap` graph, except that instead of caps, symbols are placed at the top and bottom of each spike. We can choose among the symbols we would use for a scatterplot. *Uses spjanfeb2001.dta*



```
twoway rbar high low tradeday
```

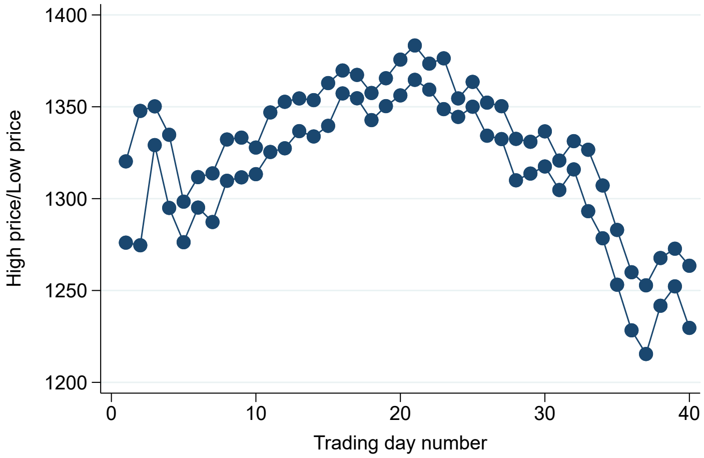
The `rbar` graph uses bars for each value of x to show the high and low values of y. *Uses spjanfeb2001.dta*



Let's now consider options to use with the `rconnected`, `rscatter`, `rline`, and `rarea` graphs. We will start by looking at `rconnected` plots because many of the options used in that kind of graph also apply to `rscatter`, `rline`, and `rarea` graphs.

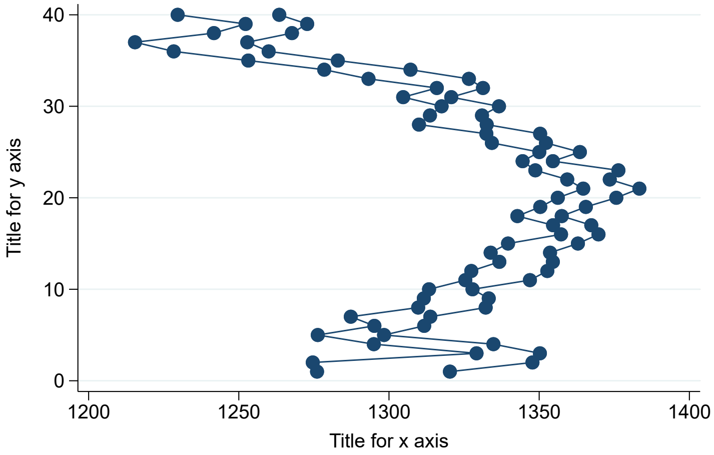
```
twoway rconnected high low tradeday, sort
```

Here is a general `rconnected` graph. *Uses spjanfeb2001.dta*



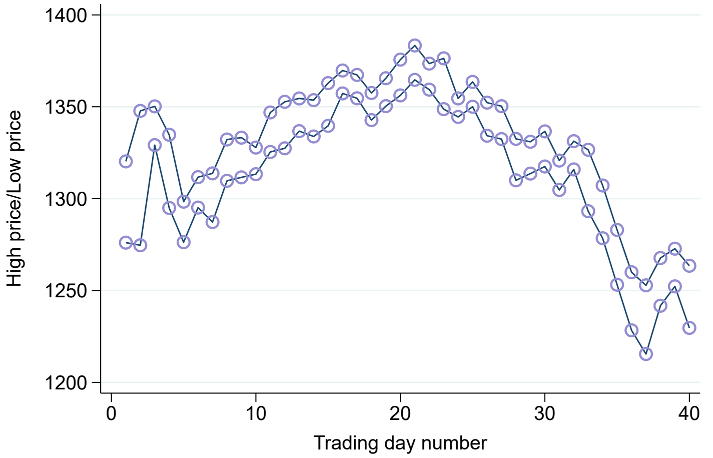
```
twoway rconnected high low tradeday, sort horizontal
xtitle(Title for x axis) ytitle(Title for y axis)
```

With the `horizontal` option, we can swap the axes where `high/low` and `tradeday` appear. The x axis remains at the bottom, and the y axis remains at the left. *Uses spjanfeb2001.dta*



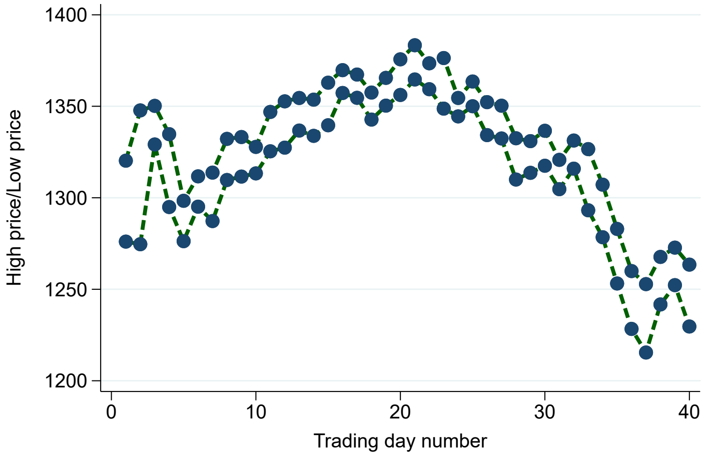
```
twoway rconnected high low tradeday, sort msymbol(Oh)
msize(large) mcolor(lavender)
```

We can control the look of the marker symbols with options such as `msymbol()`, `msize()`, and `mcolor()`. Here we make the marker symbols large, lavender, hollow circles. See Options : Markers (section 8.1) for more details. *Uses spjanfeb2001.dta*

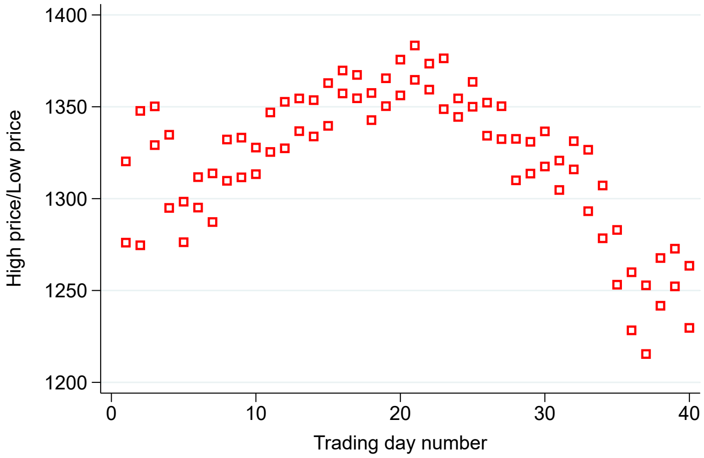


```
twoway rconnected high low tradeday, sort lwidth(thick)
lcolor(dkgreen) lpattern(dash)
```

Here we use the `lwidth()`, `lcolor()`, and `lpattern()` options to make the lines thick, dark green, and dashed. See Options : Connecting (section 8.3) for more examples, and see Styles : Connect (section 10.5), Styles : Linewidth (section 10.7), Styles : Colors (section 10.2), and Styles : Linepatterns (section 10.6) for more details about the mentioned options. *Uses spjanfeb2001.dta*



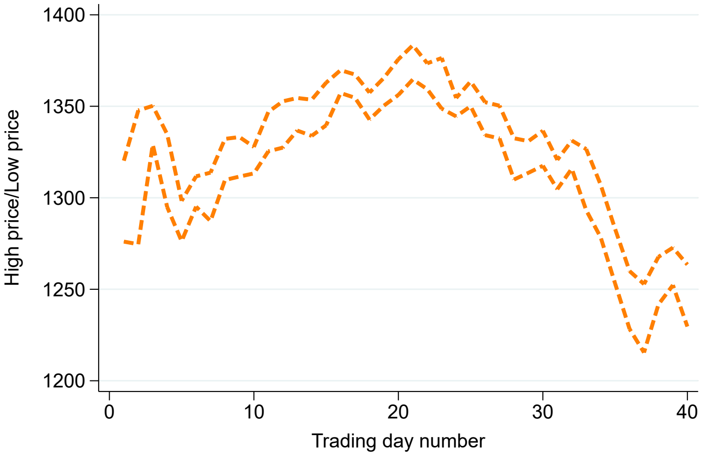
```
twoway rscatter high low tradeday, sort mcolor(red)
msize(medium) msymbol(Sh)
```



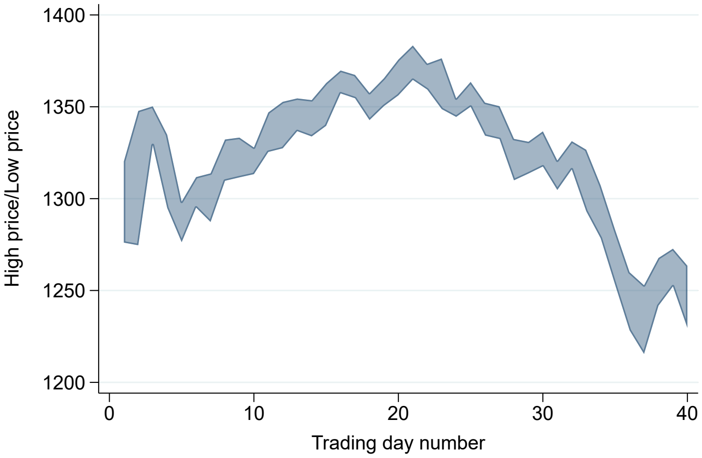
With `rscatter`, we can use the options that modify the markers. Here we change the color to red, the marker size to medium, and the markers to hollow squares. For more details about options related to marker symbols, see **Options : Markers** (section 8.1). Also see **Styles : Colors** (section 10.2) for more details about colors (including intensity and opacity). *Uses spjanfeb2001.dta*

```
twoway rline high low tradeday, sort lcolor(orange)
lwidth(thick) lpattern(dash)
```

With `rline`, we can use the options that modify the lines to make them orange, thick, and dashed. For more details about connect options, see **Options : Connecting** (section 8.3). *Uses spjanfeb2001.dta*



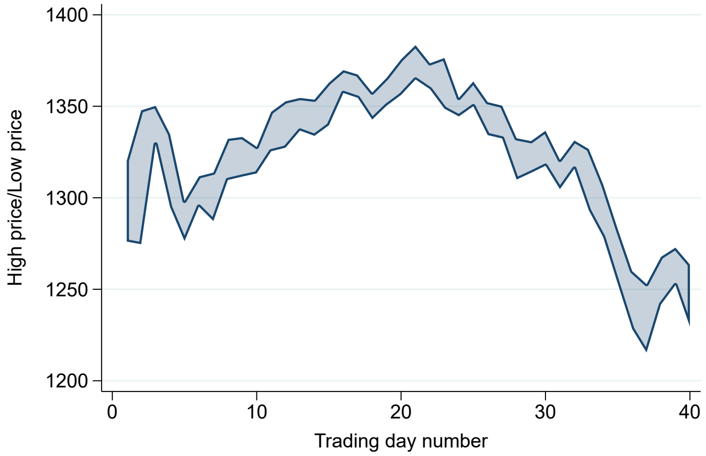
```
twoway rarea high low tradeday, sort bcolor(navy%50)
```



The `rarea` graph is similar to the `rline` graph, but in addition to being able to control the characteristics of the line, we can control the color of the area between the low and high lines. Here we use the `bcolor()` option to make the color of the line and the area `dknavy%50`, dark navy with 50% opacity. See **Styles : Colors** (section 10.2) for more details about colors (including intensity and opacity). *Uses spjanfeb2001.dta*

```
twoway rarea high low tradeday, sort fcolor(navy%30)
lcolor(navy) lwidth(medthick)
```

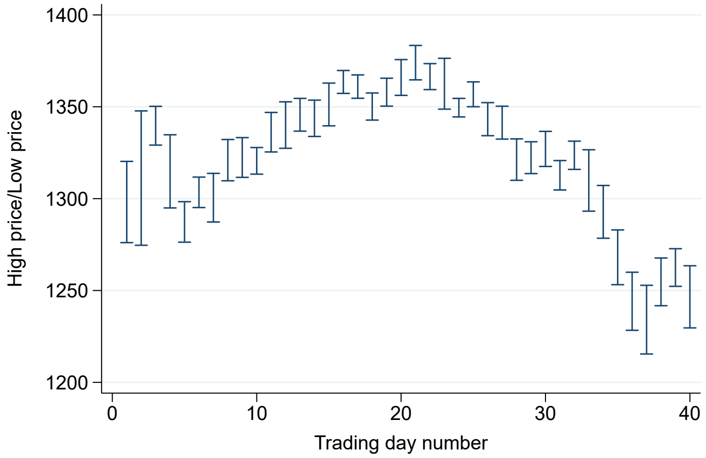
In this example, the fill color is `navy` with 30% opacity, the outline color is `navy`, and the outline color is medium thick. See **Styles : Colors** (section 10.2) for more details about colors (including intensity and opacity). *Uses spjanfeb2001.dta*



Now let's look at options for use with the `rcap`, `rspike`, and `rcapsym` graphs. The options permitted by the `rcap` option are similar to the options used with the `rspike` and `rcapsym` graphs.

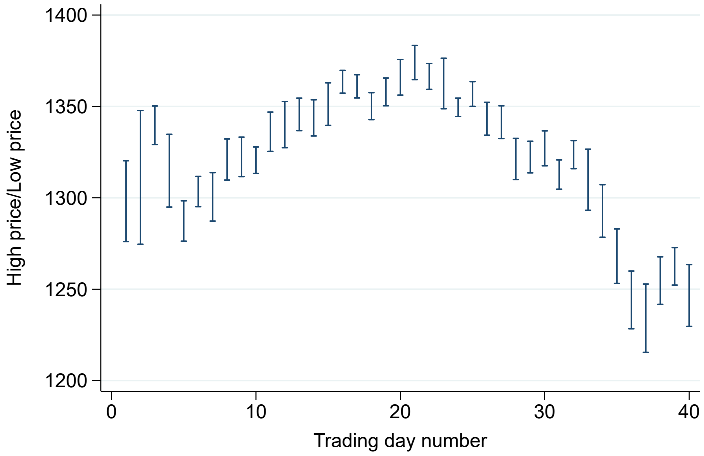
```
twoway rcap high low tradeday
```

Here is a basic `rcap` graph. The `rcap` command supports the `horizontal` option, which would make the variables `high`/`low` and `tradeday` swap positions. *Uses spjanfeb2001.dta*

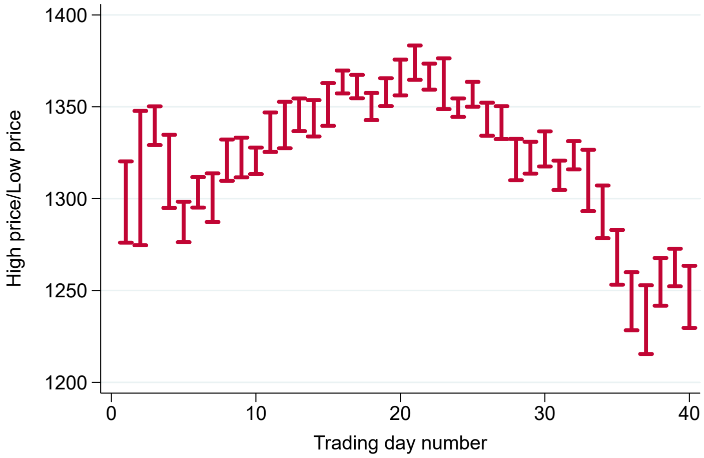


```
twoway rcap high low tradeday, msize(small)
```

We usually use the `msize()` option to control the size of a marker, and it is adapted for this kind of graph to control the size of the cap. In this example, we make the cap small. *Uses spjanfeb2001.dta*



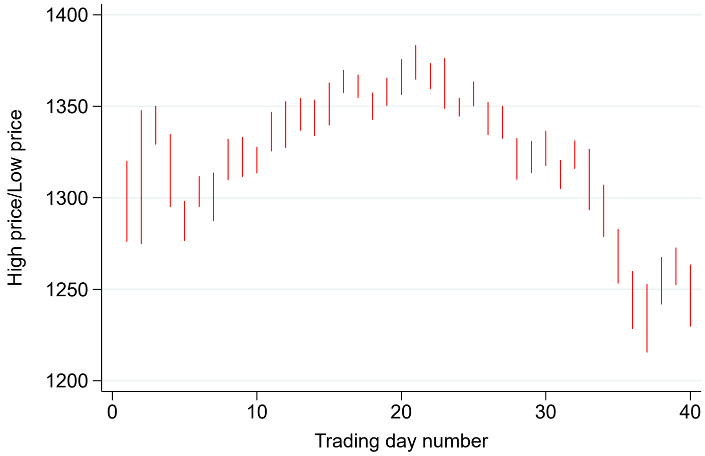
```
twoway rcap high low tradeday, lcolor(cranberry)
lwidth(thick)
```



We use the `lcolor()` option to control the color of the line, here, cranberry. We use the `lwidth()` option to set the width of the line, here, thick. Although it is not shown here, you could also control the pattern of the line with the `lpattern()` option. See Options : Connecting (section 8.3) for more details. *Uses spjanfeb2001.dta*

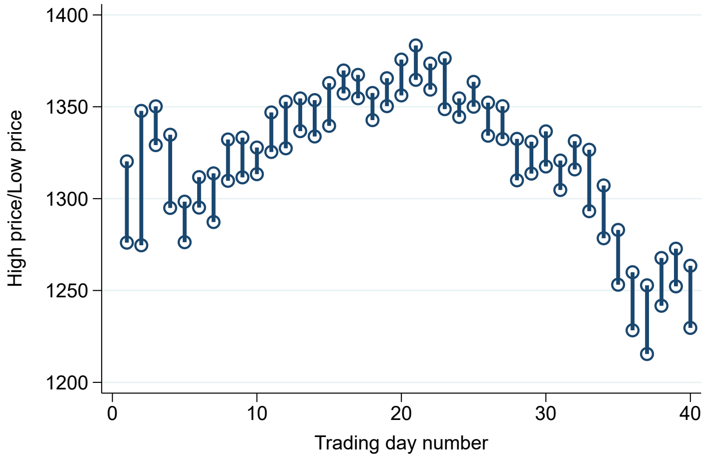
```
twoway rspike high low tradeday, lcolor(red) lwidth(thin)
```

The options used for `rspike` are basically the same as those for `rcap`, except that the `msize()` option is not appropriate because there are no markers to size. Here, for example, we use `lcolor()` and `lwidth()` to make the lines red and thin. *Uses spjanfeb2001.dta*



```
twoway rcapsym high low tradeday, msymbol(Oh) lwidth(thick)
```

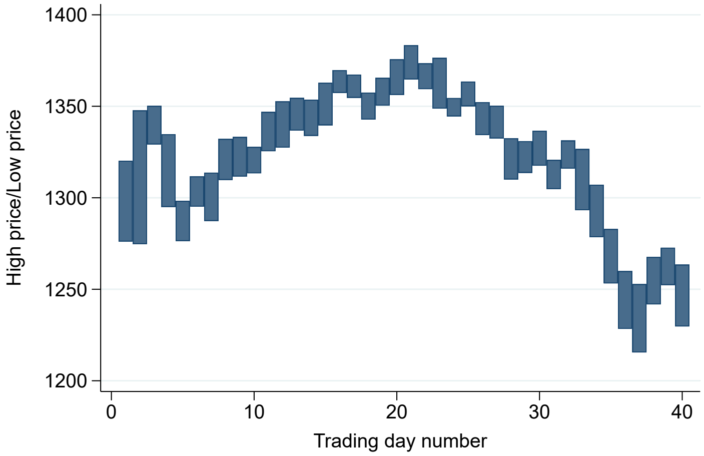
The options used for `rcapsym` allow us to modify both the markers and the lines. Here we use the `msymbol()` option to place hollow circles at the end of the spikes and the `lwidth()` option to make the lines thick. *Uses spjanfeb2001.dta*



Let’s now explore options to use with `twoway rbar`.

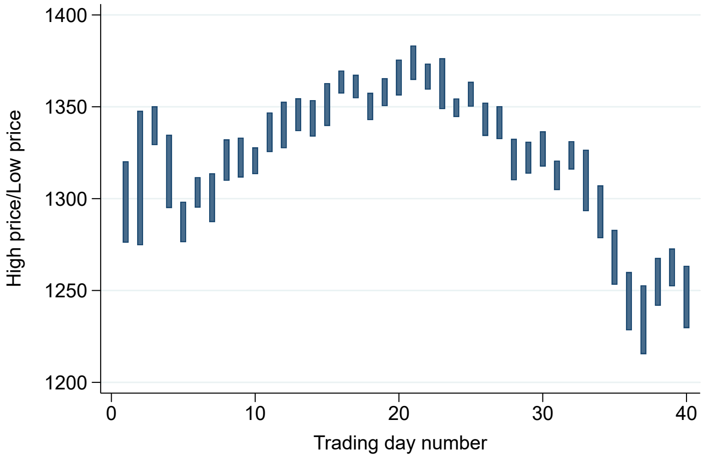
```
twoway rbar high low tradeday
```

Here is a basic `rbar` graph. As with the other graphs in this family, you could have added the `horizontal` option to switch the position of the `high/low` and `tradeday` variables, but this is not shown. *Uses spjanfeb2001.dta*



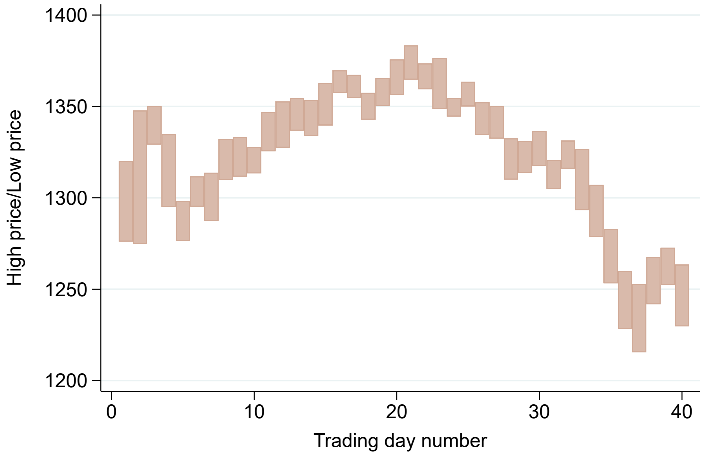
```
twoway rbar high low tradeday, barwidth(.4)
```

We can use the `barwidth()` option to set the width of the bar. This width is in units of the x variable. We set the bars to 0.4 units wide, so they no longer touch each other. *Uses spjanfeb2001.dta*



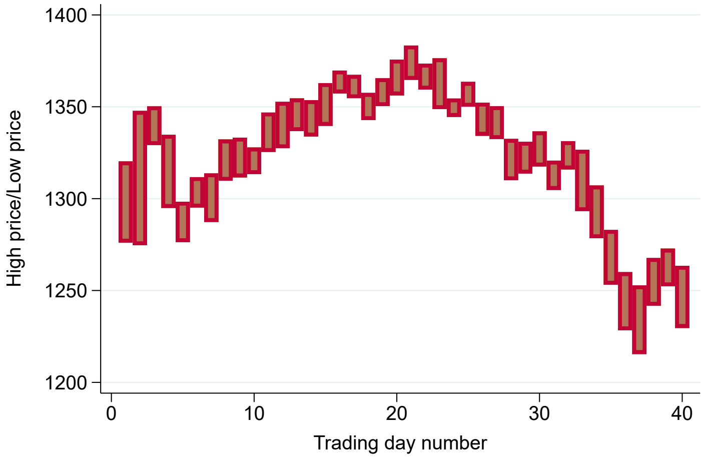
```
twoway rbar high low tradeday, bcolor(sienna*0.5)
```

The `bcolor()` (bar color) option sets the color of the bar and the outline. Here we make the color a brighter version of sienna by specifying `sienna*0.5`; See **Styles : Colors** (section 10.2) for more details about colors (including intensity and opacity). *Uses spjanfeb2001.dta*



```
twoway rbar high low tradeday, fcolor(sienna)
lcolor(cranberry) lwidth(thick)
```

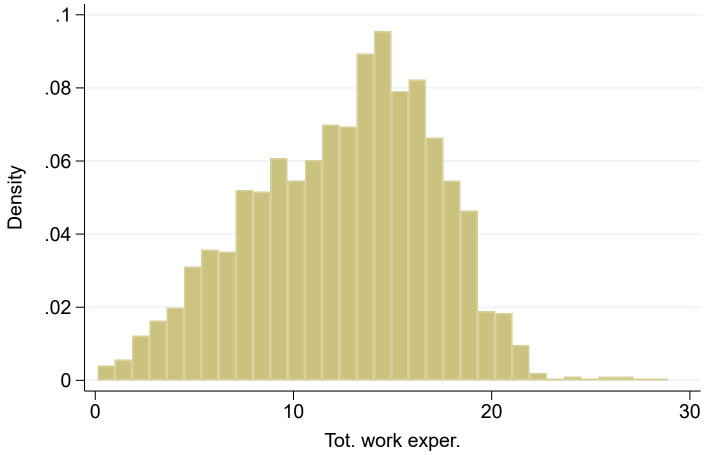
With the `fcolor()` (fill color), `lcolor()` (line color), and `lwidth()` (line width) options, we make sienna-filled bars, outlined with thick cranberry lines. *Uses spjanfeb2001.dta*



2.8 Distribution plots

This section describes the use of `twoway histogram` and `twoway kdensity` for showing the distribution of one variable. This section also shows the use of `twoway function` for showing the relationship between x and y using a function that you specify. See `help graph twoway histogram`, `help graph twoway kdensity`, and `help graph twoway function` for more information. This section starts by showing the `twoway histogram` command and then illustrates options that allow you to control such things as the number of bins, the width of the bins, and the starting point for the bins. The section concludes with options that control the scaling of the y axis.

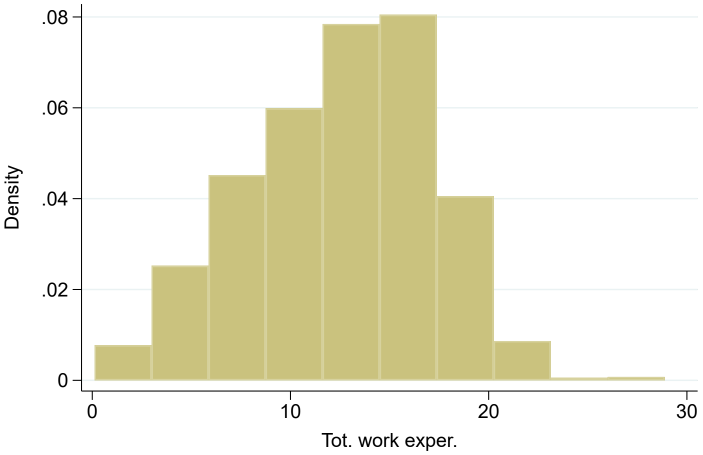
```
twoway histogram ttl_exp
```



We begin by showing a `histogram` of the variable total work experience. Unlike many other twoway plots, this command takes only one variable that is graphed on the x axis. The y axis represents the density, such that the sum of the areas of the bars equals 1. If we are not going to combine this graph with other twoway graphs, the `histogram` command may be preferable to `twoway histogram`. *Uses nlsw.dta*

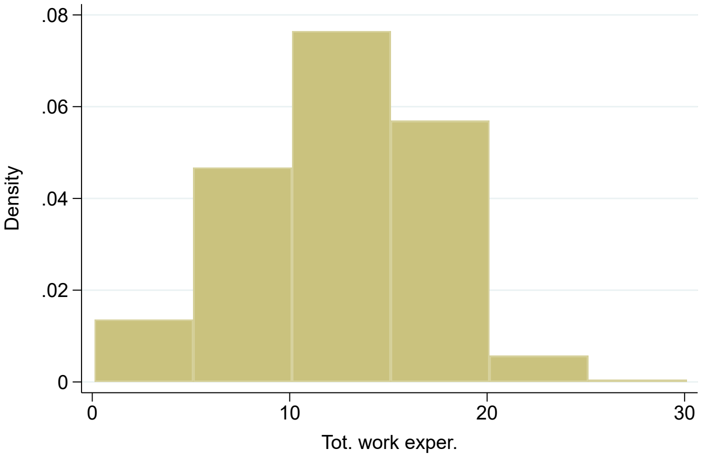
```
twoway histogram ttl_exp, bin(10)
```

We can control the number of bins that are used to display the histogram by specifying the `bin()` option. In this example, we request that 10 bins be used. *Uses nlsw.dta*



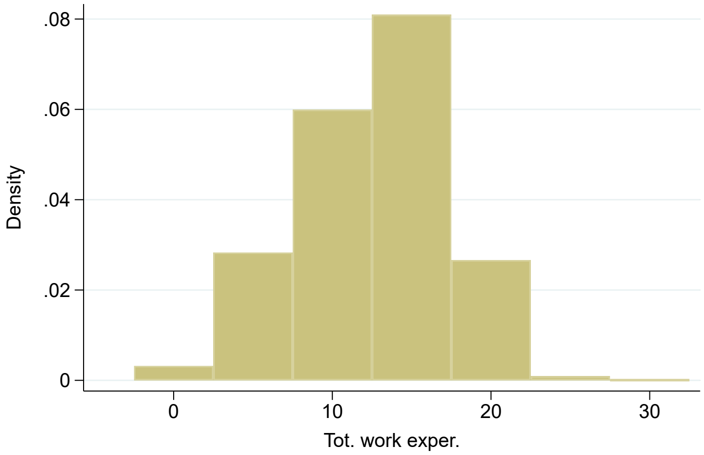
```
twoway histogram ttl_exp, width(5)
```

We can control the width of each bar using the `width()` option. Here we make each bar 5 units wide. As one might imagine, we can use either the `bin()` option or the `width()` option, but not both. *Uses nlsw.dta*



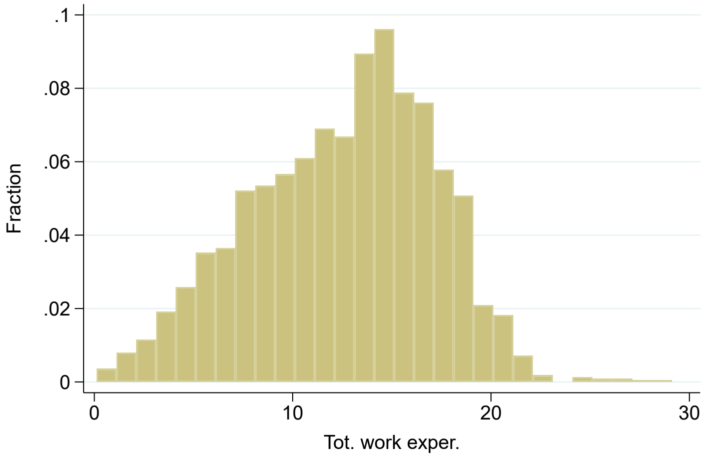
```
twoway histogram ttl_exp, start(-2.5) width(5)
```

We add the `start()` option to indicate that we want the lower limit of the first bin to start at − 2.5. *Uses nlsw.dta*



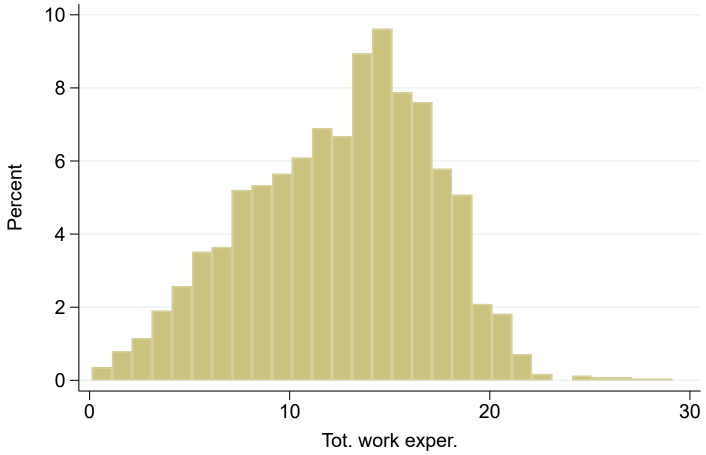
```
twoway histogram ttl_exp, fraction width(1)
```

If we use the `fraction` option, the y axis is scaled such that the height of each bar is the probability of falling within the range of x values represented by the bar. Thus, if we specify the width of bars to 1, the sum of the heights of the bars is 1. *Uses nlsw.dta*



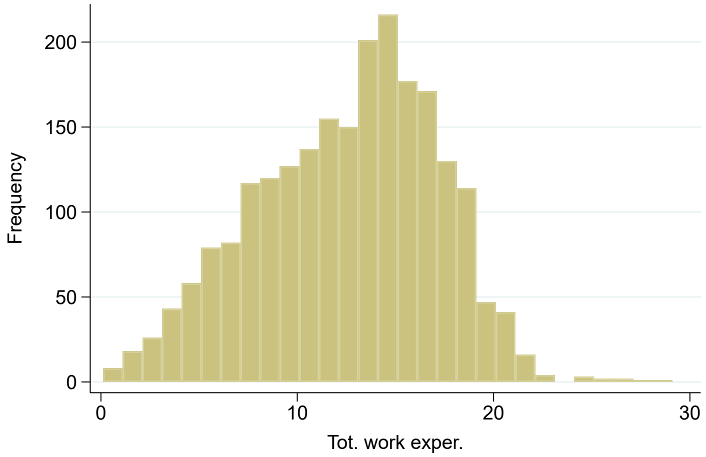
```
twoway histogram ttl_exp, percent width(1)
```

The `percent` option is similar to the `fraction` option, except that the y axis is represented as a percentage instead of a proportion. If we also specify a bar width of 1, the sum of the heights of the bars is 100%. *Uses nlsw.dta*



```
twoway histogram ttl_exp, frequency width(1)
```

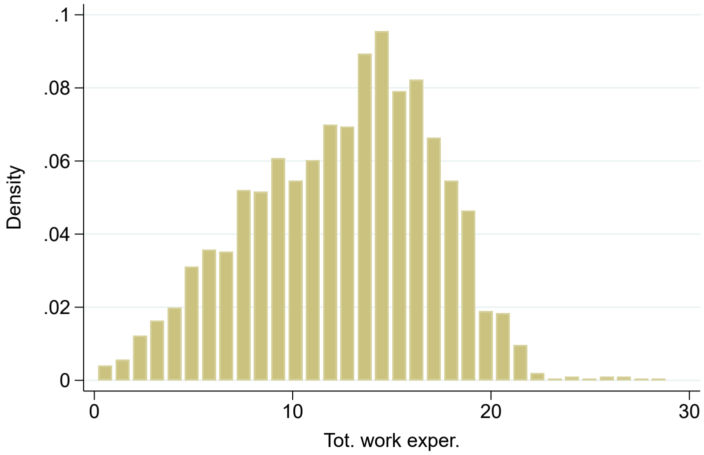
The `frequency` option changes the scaling of the y axis to represent the number of cases that fall within the range of x values represented by the bar. If we specify a bar width of 1, the sum of the heights of the bars equals the number of nonmissing values for `ttl_exp`. *Uses nlsw.dta*



Let's now consider options that control the width of the bars and other characteristics of the bars, such as color. Then we will see how to display the graph as a horizontal histogram as well as options that allow you to treat *varname* as a discrete variable.

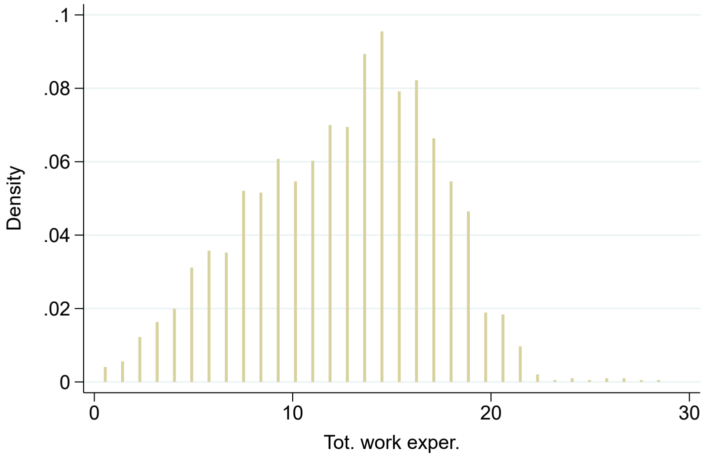
```
twoway histogram ttl_exp, gap(20)
```

The `gap()` option specifies the gap between each of the bars. The gap is created by reducing the width of the bars. By default, the gap is 0, meaning that the bars touch exactly and the bars are reduced by 0%. Here we reduce the size of the bars by 20%, making a small gap between the bars. *Uses nlsw.dta*



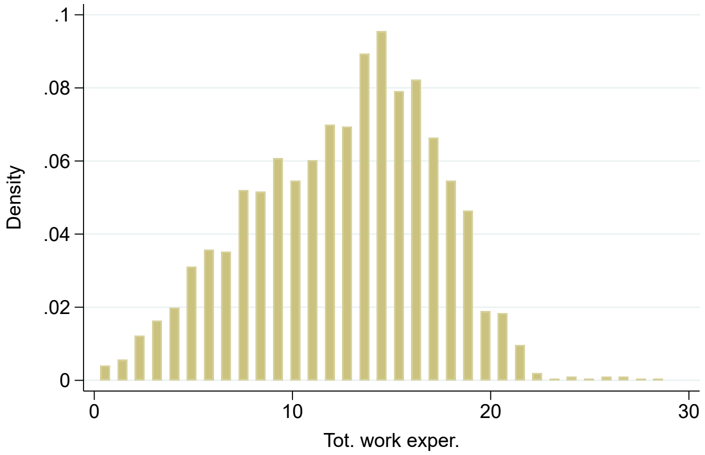
```
twoway histogram ttl_exp, gap(99.99)
```

In this example, we reduce the size of the bars 99.99%, making the bars 0.01% of their normal size. *Uses nlsw.dta*

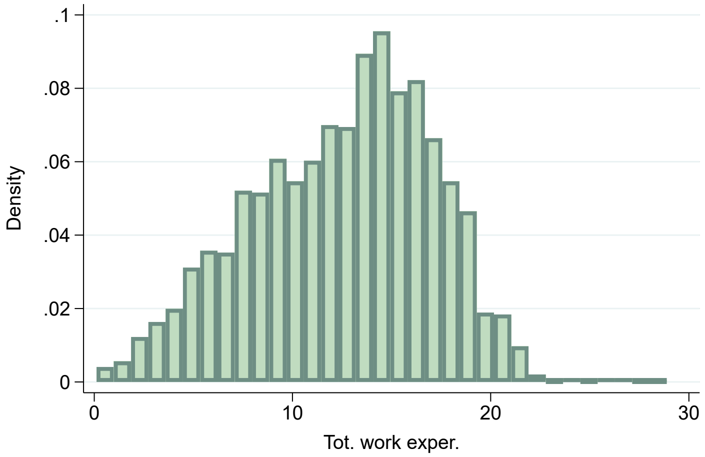


```
twoway histogram ttl_exp, barwidth(.5)
```

Another way you can control the width of the bars is with the `barwidth()` option. Here we indicate that we want each bar 0.5 x-units wide. *Uses nlsw.dta*



```
twoway histogram ttl_exp, fcolor(olive_teal) lcolor(teal)
lwidth(thick)
```

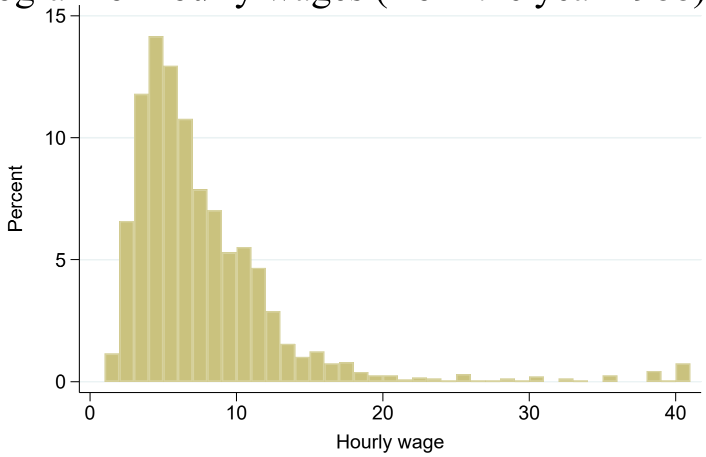


The `fcolor()` (fill color) option makes the fill color of the bar olive-teal, and the `lcolor()` (line color) option makes the bar line color teal. The `lwidth()` (line width) option makes the line around the bar thick. The section **Styles : Colors** (section 10.2) shows more about colors, and **Styles : Linewidth** (section 10.7) shows more about line widths. *Uses nlsw.dta*

This section will illustrate how to create histograms that overlap. A key part of this discussion will involve the overlapping of colors, including adjusting opacity to facilitate the display of overlap. For more details on overlapping colors, see section 10.2.4.

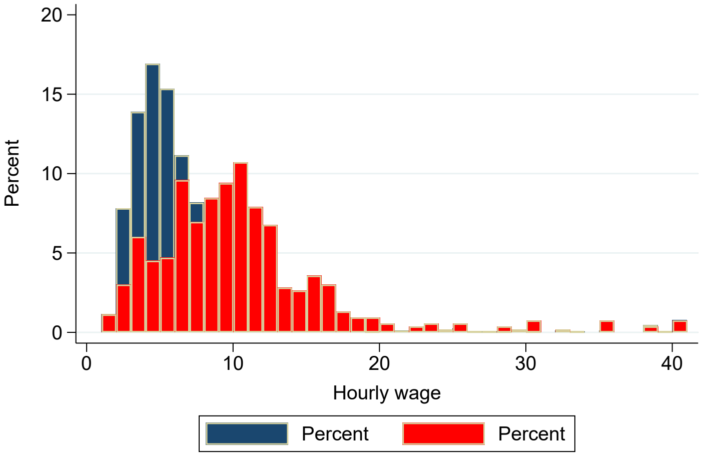
```
twoway histogram wage, width(1) percent
```

This first example shows a histogram of hourly wages (from the year 1988). We include the `percent` option, so the height of each bar represents a percentage of all observations earning the wages represented by the bar. Using the `width(1)` option, we show that each bar represents a width of 1 dollar in wages. *Uses nlsw.dta*



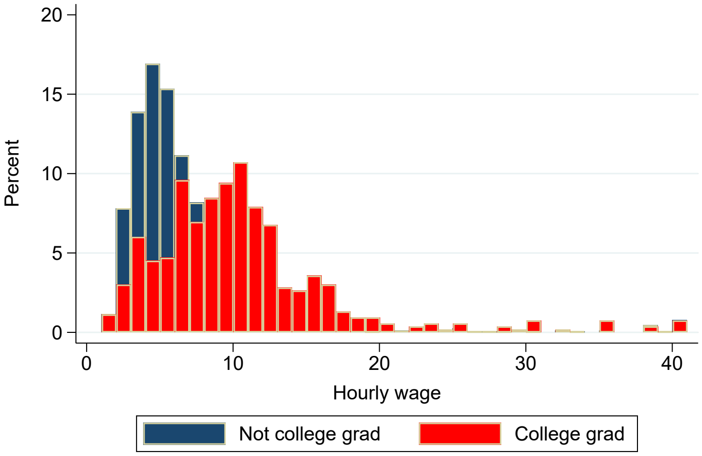
```
twoway (histogram wage if collgrad==0, width(1) percent
fcolor(navy)) (histogram wage if collgrad==1, width(1)
percent fcolor(red))
```

This graph shows the distribution of hourly wages separately by college graduation status. The first plot uses the `histogram` command with `if collgrad==0`, while the second plot uses the `histogram` command with `if collgrad==1`. The legend is not well labeled, and the red bars for the college graduates obscure the blue bars for the noncollege graduates. *Uses nlsw.dta*



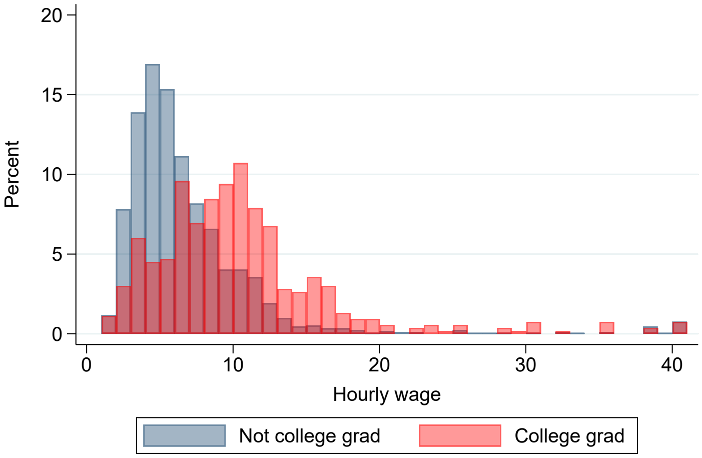
```
twoway (histogram wage if collgrad==0, width(1) percent
fcolor(navy)) (histogram wage if collgrad==1, width(1)
percent fcolor(red)), legend(label(1 "Not college grad")
label(2 "College grad"))
```

This graph is improved by the addition of the `legend()` option, but the red bars for the college graduates still obscure the blue bars for the noncollege graduates. *Uses nlsw.dta*



```
twoway (histogram wage if collgrad==0, width(1) percent
color(navy%40)) (histogram wage if collgrad==1, width(1)
percent color(red%40)), legend(label(1 "Not college grad")
label(2 "College grad"))
```

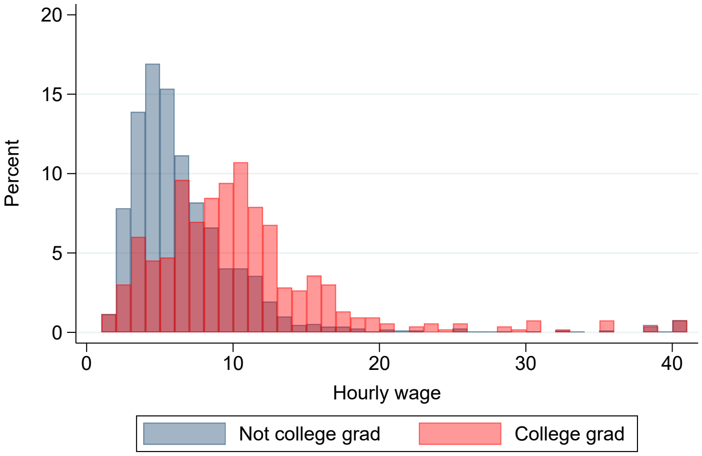
In **Styles : Colors** (section 10.2), you can see how to control the opacity (the inverse of transparency) of colors. The fill color for the noncollege graduates is shown using `navy` with 40% opacity via the `color(navy%40)` option; for college graduates, the fill color is specified as `red` with 40% via the `color(navy%40)` option. *Uses nlsw.dta*



The following commands will use abbreviations to make the command fit on the page. Specifically, `twoway` will be abbreviated as `two`, `histogram` as `hist`, `width(1)` as `w(1)`, `lwidth()` as `lw()`, and `color()` as `col()`. The text will refer to the full name, even if the command or option has been abbreviated.

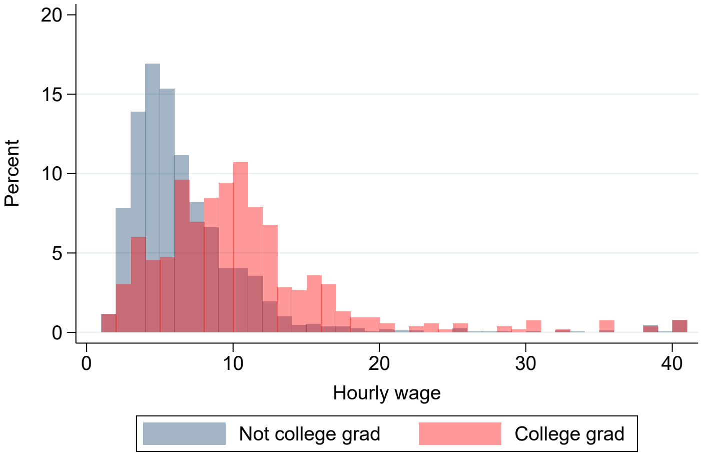
```
two (hist wage if collgrad==0, w(1) percent fcol(navy%40)
lcol(navy%40) lw(vthin)) (hist wage if collgrad==1, w(1)
percent fcol(red%40) lcol(red%40) lw(vthin)), legend(label(1
"Not college grad") label(2 "College grad"))
```

We add the `lwidth(vthin)` option to each `histogram`, making the line width very thin. (See **Styles : Linewidth** (section 10.7) for more options on specifying line widths, including the ability to specify widths in points.) I prefer the look of this graph that uses very thin lines.
Uses nlsw.dta

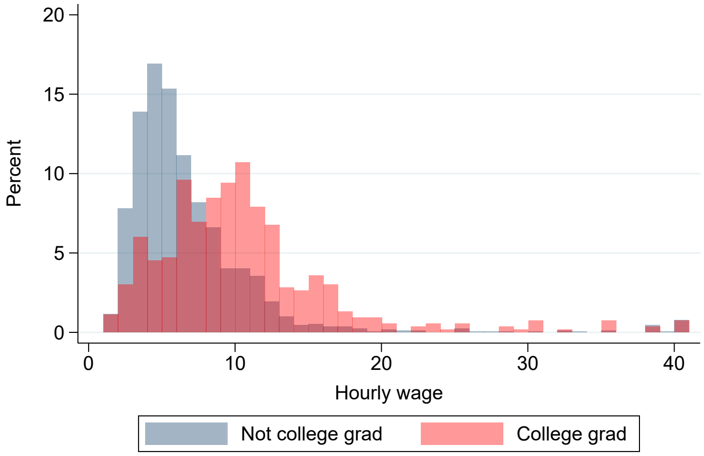


```
two (hist wage if collgrad==0, w(1) percent fcol(navy%40)
lw(none)) (hist wage if collgrad==1, w(1) percent
fcol(red%40) lw(none)), legend(label(1 "Not college grad")
label(2 "College grad"))
```

In this example, we specify `lw(none)` to eliminate the line surrounding the bars. As illustrated in **Styles : Linewidth** (section 10.7), when we specify a line width of `none`, the line has zero width; it vanishes. Thus, the `lcolor()` options are now superfluous and could have been removed.
Uses nlsw.dta



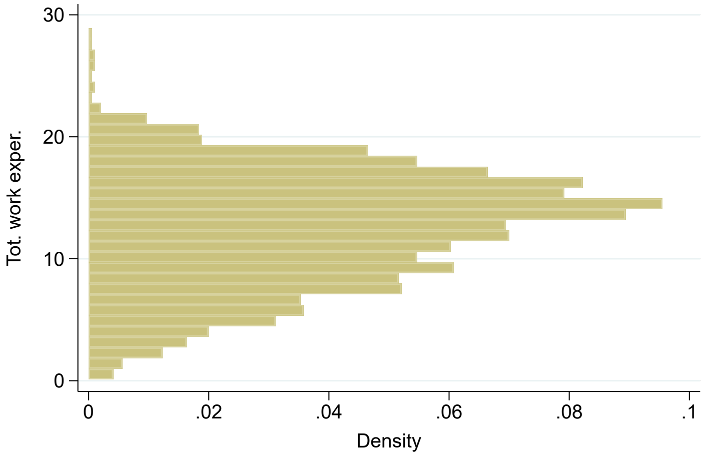
```
two (hist wage if collgrad==0, w(1) percent fcol(navy%40)
lcol(none)) (hist wage if collgrad==1, w(1) percent
fcol(red%40) lcol(none)), legend(label(1 "Not college grad")
label(2 "College grad"))
```



The command in this example produces the same graph as shown in the prior example but uses a simpler method. When we specify `lcolor(none)` for each `histogram` command, each line surrounding the histogram has *no color*, meaning it is invisible. No line is drawn. *Uses nlsw.dta*

Let's now briefly consider other options that can be used with `twoway histogram`, seeing how to swap the position of the x and y axes, and the `discrete` option for use with discrete variables.

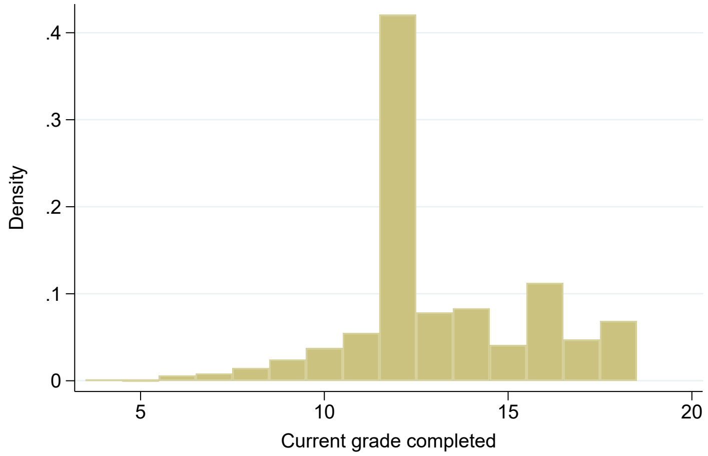
```
twoway histogram ttl_exp, horizontal
```



The `horizontal` option swaps the position of `ttl_exp` and its density, making a horizontal display of the histogram. *Uses nlsw.dta*

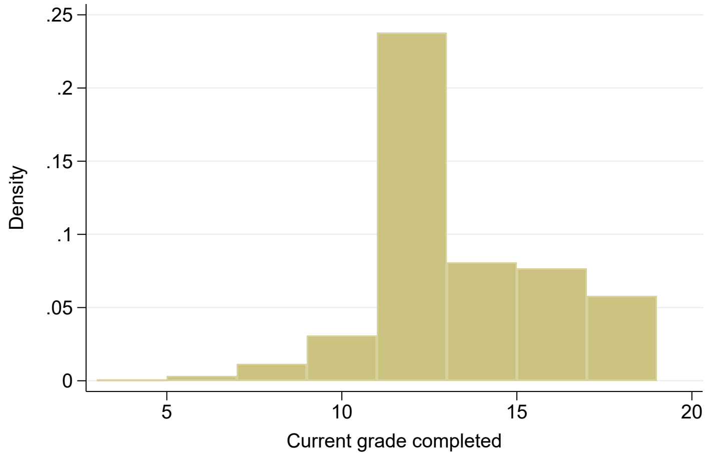
```
twoway histogram grade, discrete
```

The `discrete` option tells Stata that the variable `grade` is a discrete variable and can take only integer values. Here each bin has a width of 1. *Uses nlsw.dta*



```
twoway histogram grade, discrete width(2)
```

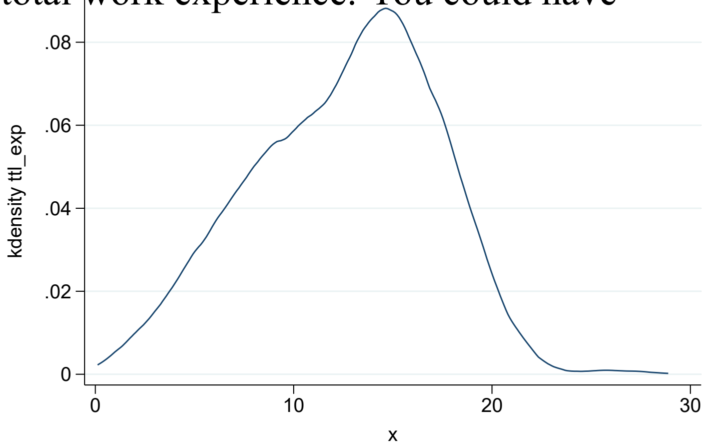
The `width()` option changes the bar's width. In this example, we change the width to 2. *Uses nlsw.dta*



This section illustrates kernel density plots created using `twoway kdensity`. For more details, see `help graph twoway kdensity`. As with histograms, if you are not going to combine the kernel density plot with other twoway plots, the `kdensity` command is more powerful (and hence preferable) to `twoway kdensity`. We will explore a handful of options that are useful for controlling the display of these graphs.

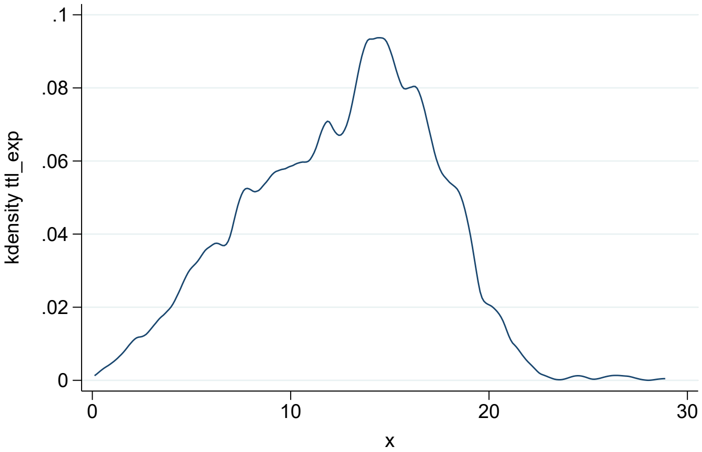
```
twoway kdensity ttl_exp
```

Here is a kernel density plot of total work experience. You could have added the `horizontal` option to display the graph as a horizontal plot, but this option is not shown. *Uses nlsw.dta*



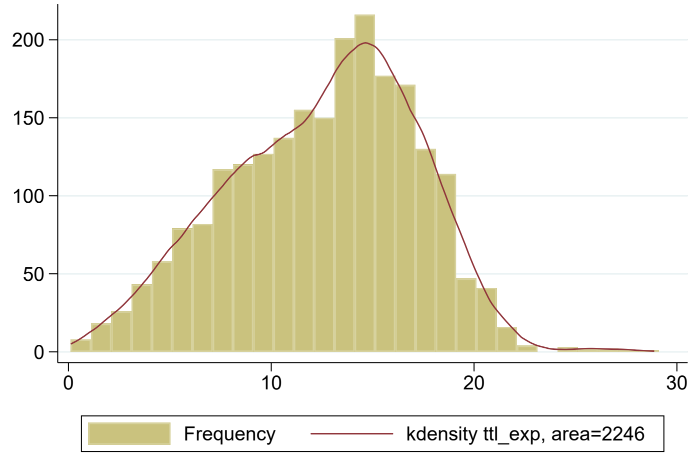
```
twoway kdensity ttl_exp, kernel(biweight)
```

By default, Stata uses an Epanechnikov kernel for computing the density estimates. Here we specify the `kernel(biweight)` option so that instead the biweight kernel for computing the densities is used. Methods include `biweight`, `cosine`, `gaussian`, `parzen`, `rectangle`, and `triangle`. *Uses nlsw.dta*

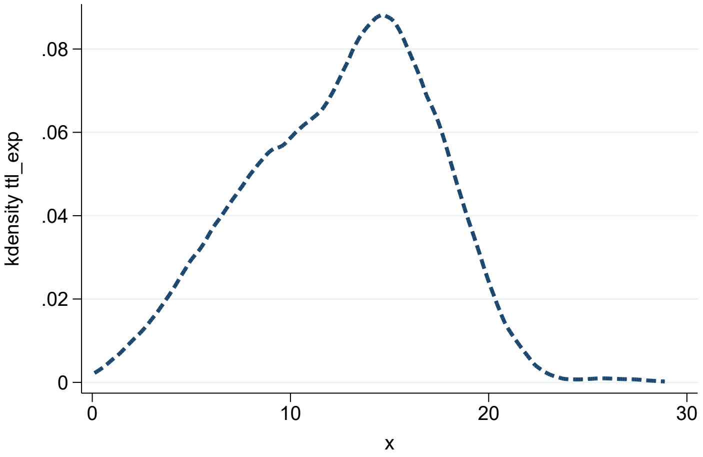


```
twoway (histogram ttl_exp, width(1) frequency) (kdensity
ttl_exp, area(2246))
```

In this example, we overlay a histogram of `ttl_exp`, scaling the y axis as the frequency of values in each bin. We overlay this with a `kdensity` plot but want to scale the y axis in a commensurate manner. By using the `area()` option, we can specify that the sum of the area of the kernel density should sum to 2,246, the sample size. *Uses nlsw.dta*



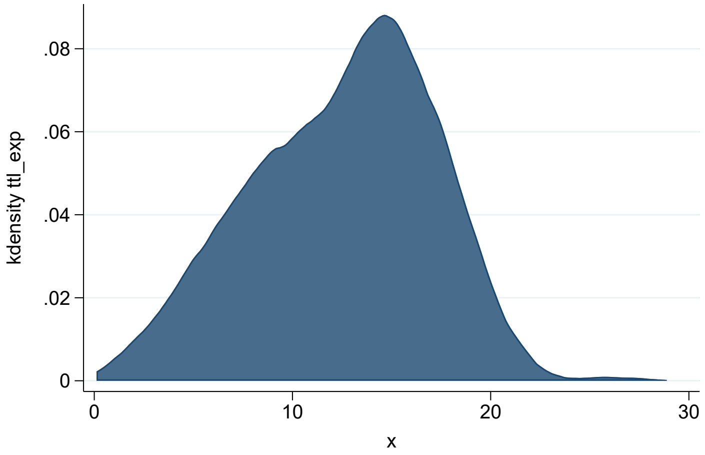
```
twoway kdensity ttl_exp, lwidth(thick) lpattern(dash)
```



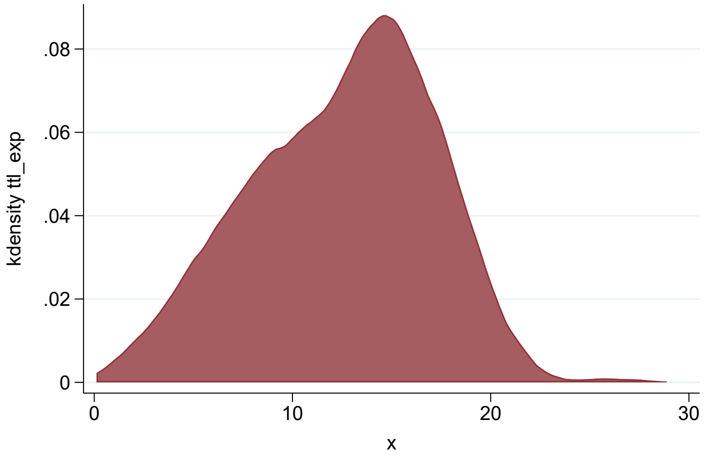
We can use the options `lcolor()`, `lwidth()`, and `lpattern()` to alter the characteristics of the line. Here we use the `lwidth()` and `lpattern()` options to make the line thick and dashed. See Styles : Linewidth (section 10.7), Styles : Linepatterns (section 10.6), and Styles : Colors (section 10.2) for more details. *Uses nlsw.dta*

```
twoway kdensity ttl_exp, recast(area)
```

We can shade the area underneath the line, like an `area` graph, by specifying `recast(area)`. See `help advanced_options` for more information about advanced `graph twoway` options, including the `recast()` option. *Uses nlsw.dta*



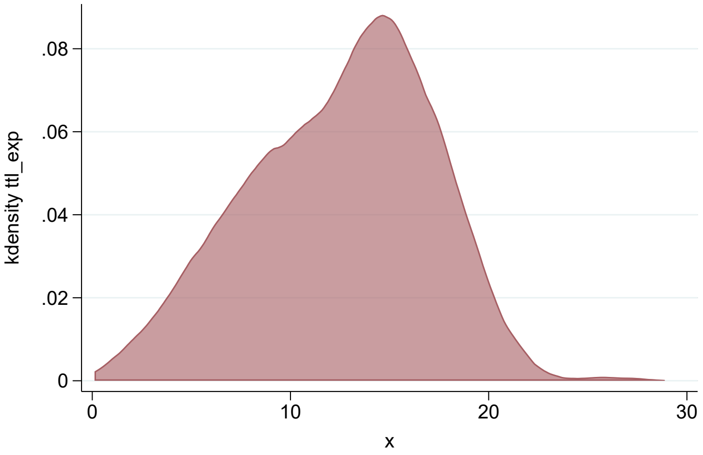
```
twoway kdensity ttl_exp, recast(area) bcolor(maroon)
```



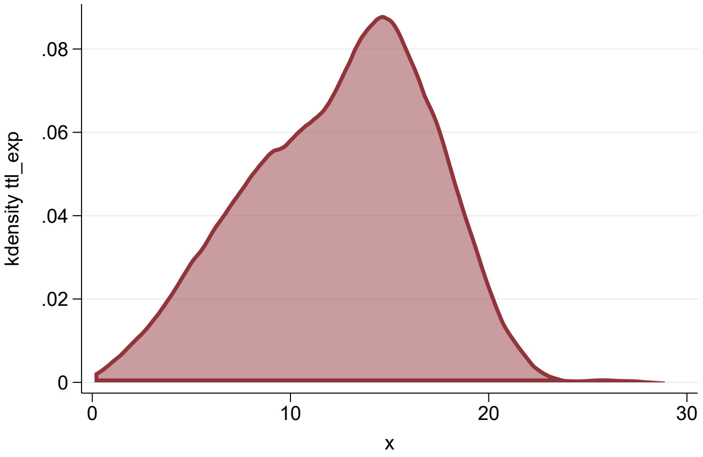
By specifying `recast(area)`, we can not only display the graph with the look of a `twoway area` graph but also specify options allowed by the `twoway area` command. As illustrated in Twoway : Area (section 2.5), we can use the `bcolor()` option to control the color of the shaded area and the line. In this example, we specify `bcolor(maroon)` to make the shaded area (and line) display as `maroon`. *Uses nlsw.dta*

```
twoway kdensity ttl_exp, recast(area) bcolor(maroon%60)
```

I would prefer to display this graph with reduced opacity for the shaded area. This example specifies `bcolor(maroon%60)` to display the shaded area with 60% opacity. See Styles : Colors (section 10.2) for more details on specifying colors. *Uses nlsw.dta*



```
twoway kdensity ttl_exp, recast(area) fcolor(maroon%60)
lcolor(maroon) lwidth(thick)
```

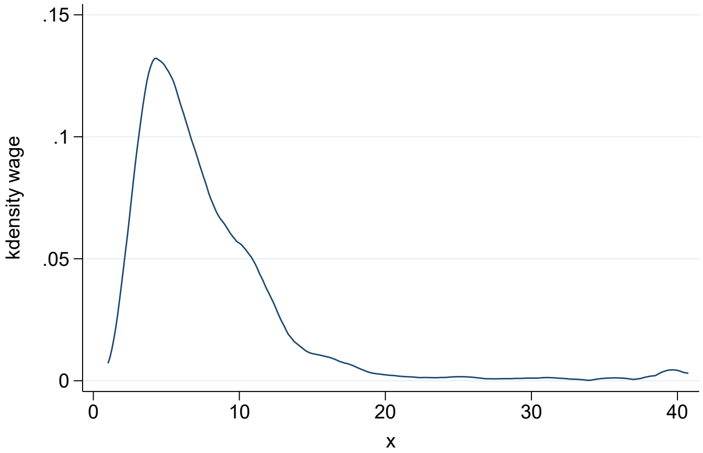


This example fills the shaded area with `maroon` with 60% opacity via the `fcolor(maroon%60)` option. The surrounding line is displayed using a maroon line with no opacity, which is due to the `lcolor(maroon)` option, and that line is thick because of the `lwidth(thick)` option. See `help area_options` for more details on options you can use with the `twoway area` command. *Uses nlsw.dta*

This section will illustrate how to create kernel density plots that overlap using the `twoway kdensity` command. A key part of this discussion will involve the overlapping of colors, including adjusting opacity to facilitate the display of overlap. For more details on overlapping colors, see section 10.2.4. Additionally, these examples will draw upon techniques that were illustrated when graphing overlapping histograms; see section 2.8 .

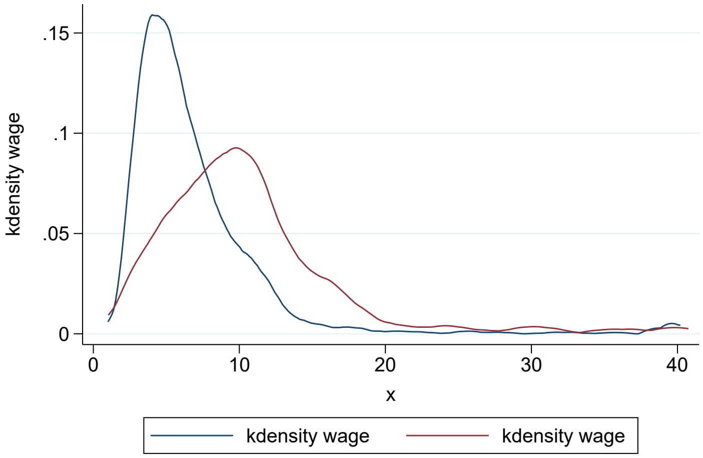
```
twoway kdensity wage
```

Consider this kernel density plot of wages (from the year 1988). It would be interesting to see how the distribution of wages differed for those who were college graduates versus those who were not college graduates. *Uses nlsw.dta*



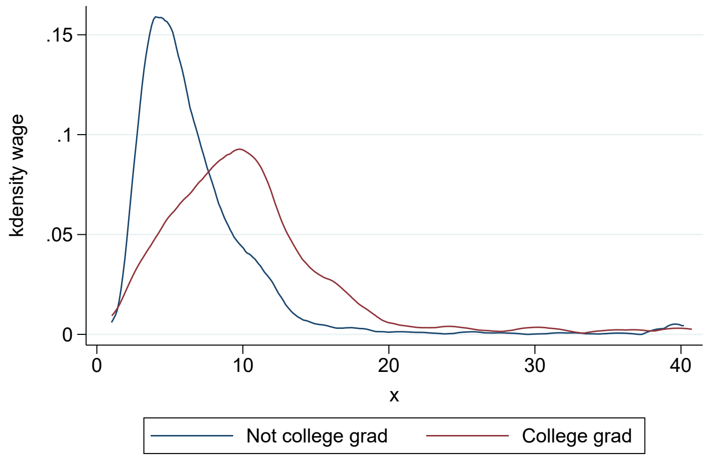
```
twoway (kdensity wage if collgrad==0) (kdensity wage if
collgrad==1)
```

This graph shows the kernel density plot by college graduation status. The first plot uses the `kdensity` command with the `if collgrad==0` specification; the second plot uses the `kdensity` command with the `if collgrad==1` specification. The legend is not well labeled. Let's fix that in the next example. *Uses nlsw.dta*

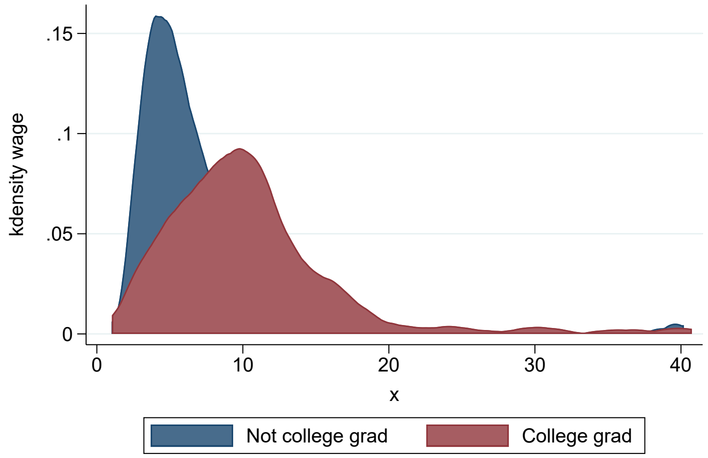


```
twoway (kdensity wage if collgrad==0) (kdensity wage if
collgrad==1), legend(label(1 "Not college grad") label(2
"College grad"))
```

This graph is improved by the addition of the `legend()` option. The graph would be more appealing if the area under the line was shaded. *Uses* *nlsw.dta*



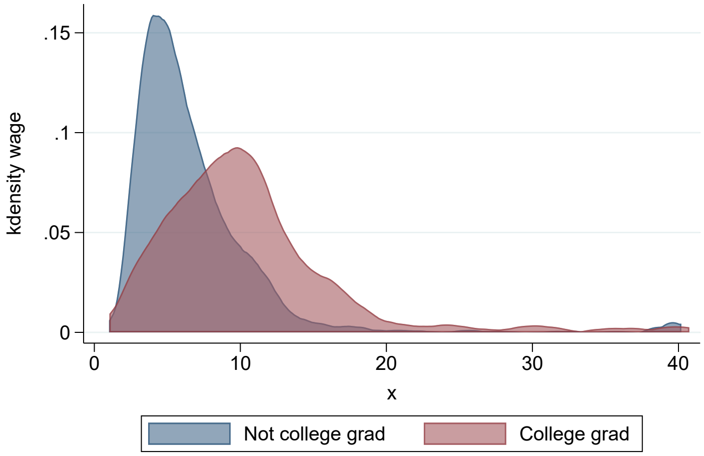
```
twoway (kdensity wage if collgrad==0, recast(area)) (kdensity
wage if collgrad==1, recast(area)), legend(label(1 "Not
college grad") label(2 "College grad"))
```



Using what we learned in the examples starting in section 2.8 , we add the `recast(area)` option to each `kdensity` command, and the area under each curve is shaded like a `twoway area` plot. The noncollege graduates are shown in `navy`, and the college graduates are shown in `maroon`. However, distribution for the college graduates conceals part of the distribution for the noncollege graduates. *Uses nlsw.dta*

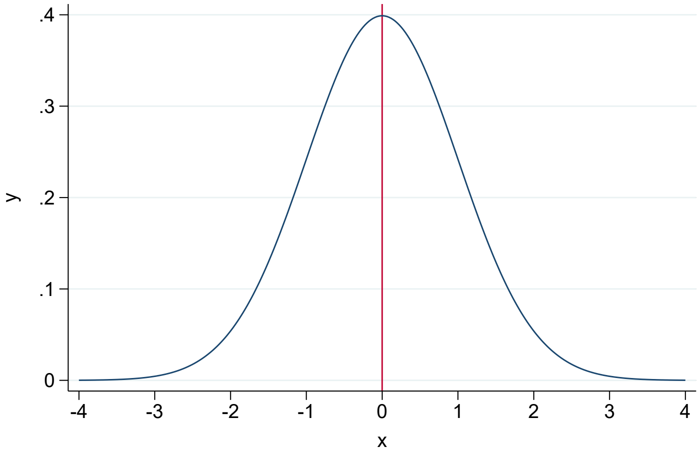
```
twoway (kdensity wage if collgrad==0, recast(area)
bcolor(%60)) (kdensity wage if collgrad==1, recast(area)
bcolor(%60)), legend(label(1 "Not college grad") label(2
"College grad"))
```

By specifying `recast(area)`, we can not only display the graph with the look of a `twoway area` graph but also specify options allowed by the `twoway area` command. As illustrated in **Twoway : Area** (section 2.5), the `bcolor()` option controls area and outline color. In this example, we specify the `bcolor(%60)` option on each `kdensity` command to display the shaded area (and line) using the default color, with 60% opacity. *Uses nlsw.dta*



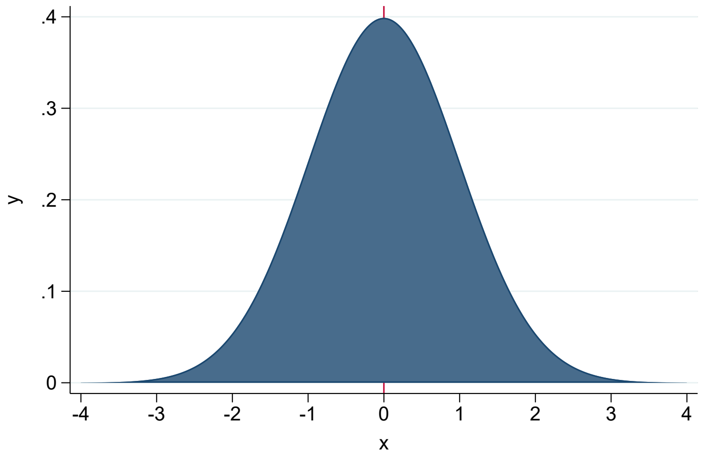
```
twoway (function y=normalden(x), range(-4 4)), xlabel(-4(1)4)
xline(0)
```

This next example illustrates the `twoway function` command, showing that we can use it to draw a standard normal curve by specifying `function y=normalden(x)`. Using the option `range(-4 4)` specifies the range of x values from -4 to 4. We add the `xlabel()` option to label x values from -4 to 4 in increments of 1 and the `xline()` option to draw an x line where x equals 0. *Uses nlsw.dta*



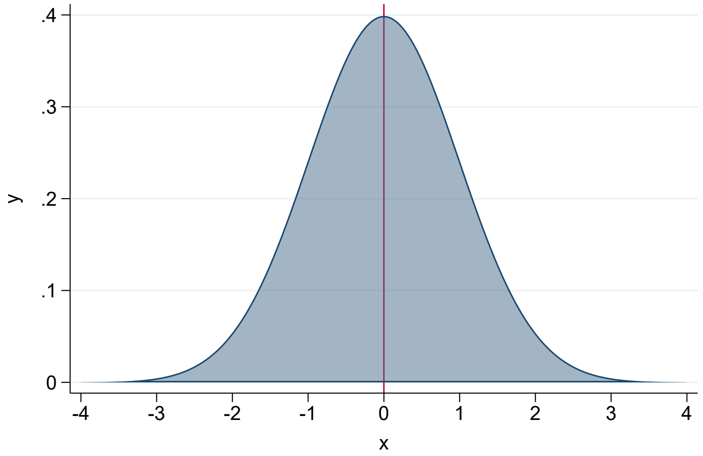
```
twoway (function y=normalden(x), range(-4 4) recast(area)),
xlabel(-4(1)4) xline(0)
```

As we saw in a recent example using the `twoway kdensity` command, we can apply the `recast(area)` option to shade the area underneath the normal curve. The x line is now obscured because it was drawn first followed by the normal curve (with its shading). We can decrease the opacity of the shading to allow the x line to show through. *Uses nlsw.dta*



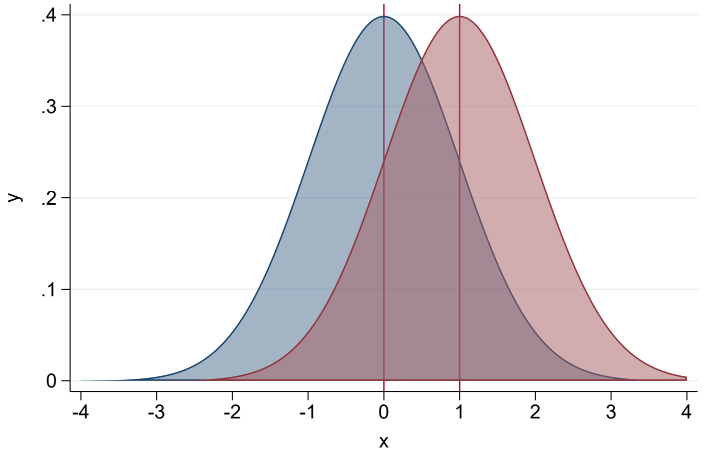
```
twoway (function y=normalden(x), range(-4 4) recast(area)
fcolor(%50)), xlabel(-4(1)4) xline(0)
```

By adding the `fcolor(%50)` option, we display the fill color with 50% opacity (in other words, with a high degree of transparency). See Styles : Colors (section 10.2) for more options you can use for specifying colors. *Uses nlsw.dta*



```
twoway (function y=normalden(x), range(-4 4) recast(area)
fcolor(%50)) (function y=normalden(x-1), range(-4 4)
recast(area) fcolor(%50)), xlabel(-4(1)4) xline(0 1)
legend(off)
```

This example adds a second `twoway function` plot. By using the `fcolor(%50)` option on the first and second `function` commands, we show both the shaded areas using their default colors with 50% opacity. The `xline(0 1)` option shows an x line at the mean of each of the normal curves. The `legend(off)` option is used to omit display of the legend. *Uses nlsw.dta*



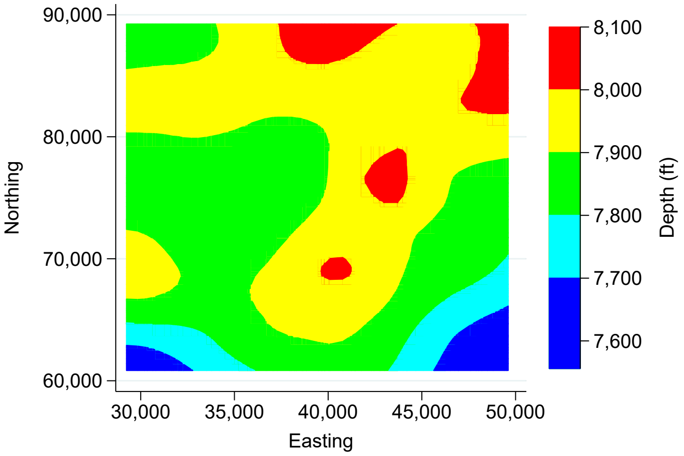
Although these last three examples focused exclusively on displaying normal curves via `twoway function` combined with the `normalden()` function, Stata offers a vast array of density functions you could use in such examples, including *chi-squared* distributions, *Poisson* distributions, *Student's t* distributions, and many others. See `help density_functions` for more information.

Further, the last example showed how you could create overlapping normal distributions. By reducing the opacity of the shaded areas, we make the graphs more clearly convey the areas of overlap in the normal curves. For more details on overlapping colors, see section 10.2.4.

2.9 Contour plots

This section focuses on the `twoway` command for creating contour plots, illustrating the `twoway contour` and `twoway contourline` commands. (For more information, see `help graph twoway contour` and `help graph twoway contourline`.) Contour plots are used in many disciplines as a means of representing three-dimensional data in two dimensions. In addition to the standard x and y axes, contour plots display a z dimension with contours that are represented using colors or lines. For example, consider `sandstone.dta` (that comes with Stata). This dataset contains geographic data regarding the depth of sandstone formations for a certain geographic region. Two dimensions are used to represent the location in the north or south direction and the east or west direction, and the third dimension indicates the depth of the sandstone layer. The following `twoway contour` command shows a contour plot for these variables using this dataset.

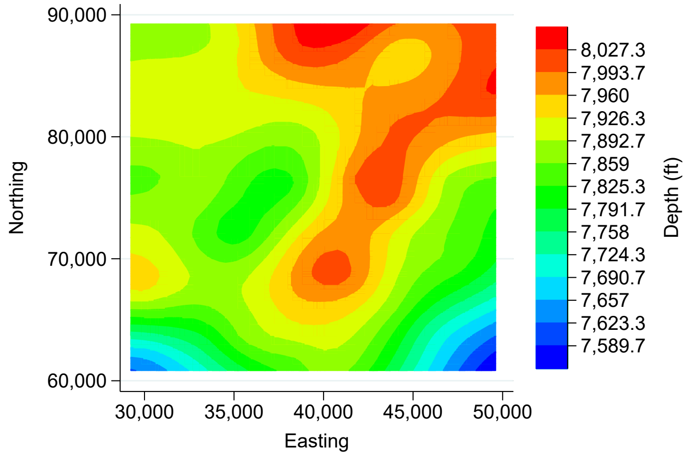
```
twoway contour depth northing easting
```



This `contour` graph shows the depth of the sandstone layers as a function of the location. The variable `depth` is graphed as the z axis, with `northing` as the y axis and `easting` as the x axis. Note how the first contour is blue, the second is cyan, the third is green, followed by yellow, and then red. The first contour color (that is, blue) represents the smallest values for `depth`, and the last contour color (that is, red) represents the highest values of `depth`. *Uses sandstone.dta*

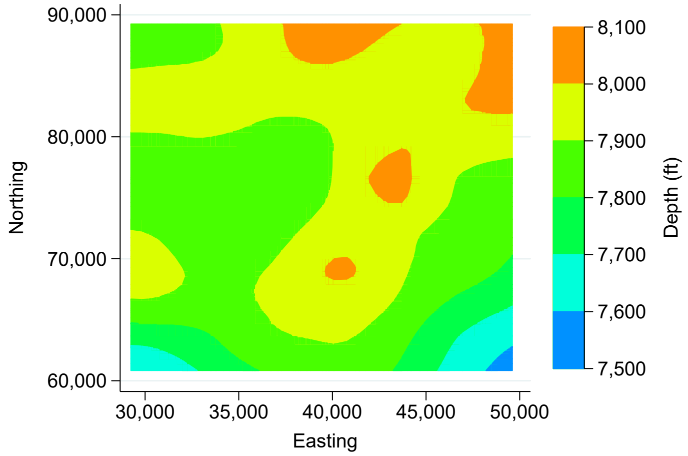
```
twoway contour depth northing easting, levels(15)
```

Note how the graph changes by adding the `levels(15)` option. Compared with the original graph that used 5 contours for `depth`, the depths are now represented using 15 contours. The key at the right shows the depth that corresponds to each of the 15 colors. *Uses sandstone.dta*



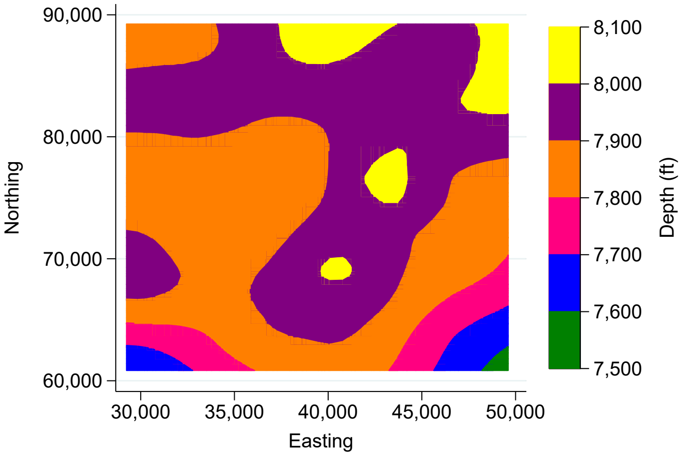
```
twoway contour depth northing easting, ccuts(7500(100)8100)
```

We can instead use the `ccuts()` option to specify the levels of the contours. In this example, we specify that the contours range from 7,500 to 8,100 in 100-foot increments. This yields a total of six contours, where blue is the smallest depth and orange is the greatest depth. Note how the colors are similar shades of green for the depths of 7,700–7,800 and 7,800–7,900. *Uses sandstone.dta*

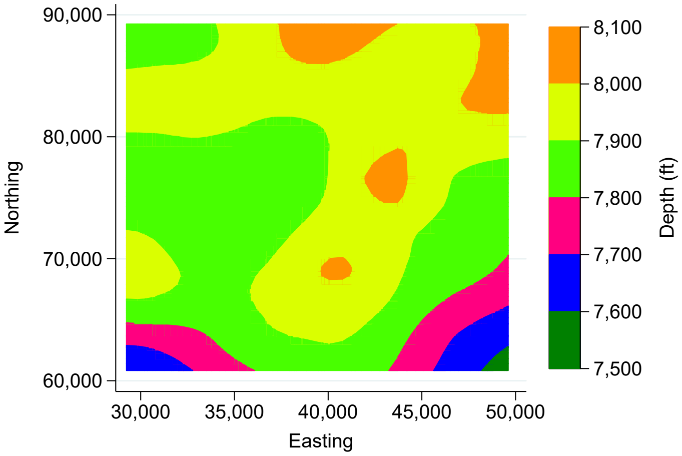


```
twoway contour depth northing easting, ccuts(7500(100)8100)
ccolors(red green blue pink orange purple yellow)
```

We can add the `ccolors()` option to specify the colors for each of the contours. Depths below 7,500 feet are colored red, 7,500–7,600 feet are green, 7,600–7,700 are blue, and so forth. The first color (red) for depths below 7,500 feet appears to be ignored, but this is because there are no observations where the depth is below 7,500 feet. *Uses sandstone.dta*



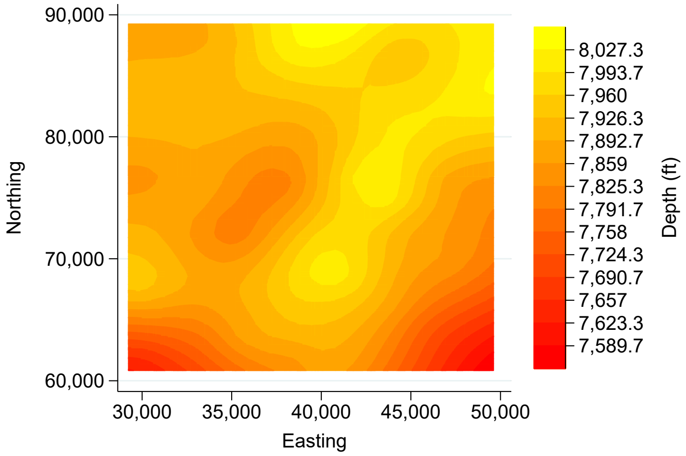
```
twoway contour depth northing easting, ccuts(7500(100)8100)
ccolors(red green blue pink)
```



Actually, we do not need to specify the colors for all the contours. In this example, we specified just four colors. The contour depths below 7,500 feet is red, for 7,500–7,600 feet is green, for 7,600–7,700 feet is blue, and for 7,700–7,800 feet is pink. The remaining contours are colored using green, yellow, and orange, the colors used when we omitted the `ccolors()` option. As in the previous example, there are no observations where the depth is below 7,500 feet. *Uses sandstone.dta*

If there is a large number of contours, it could be cumbersome to specify each color for all the contours. Instead, Stata provides you the option of specifying the starting contour color, the ending contour color, and a method for choosing the intermediate colors. This is illustrated in the following examples.

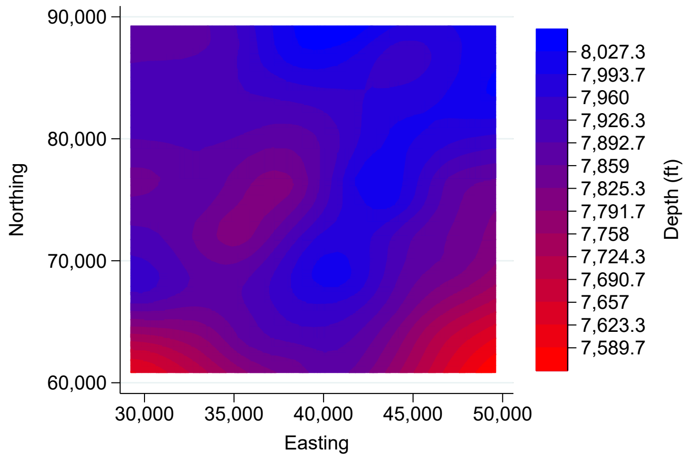
```
twoway contour depth northing easting, levels(15)
crule(linear) scolor(red) ecolor(yellow)
```



This example uses the `crule(linear)` option combined with the `scolor(red)` and `ecolor(yellow)` options. The result is that the colors start with pure red at the lowest depths and end at pure yellow at the highest depths. As depth increases, the color is a mixture of red and yellow, with increasing amounts of yellow for higher depths. The `scolor()` option specifies the starting color, and the `ecolor()` option specifies the ending color. *Uses sandstone.dta*

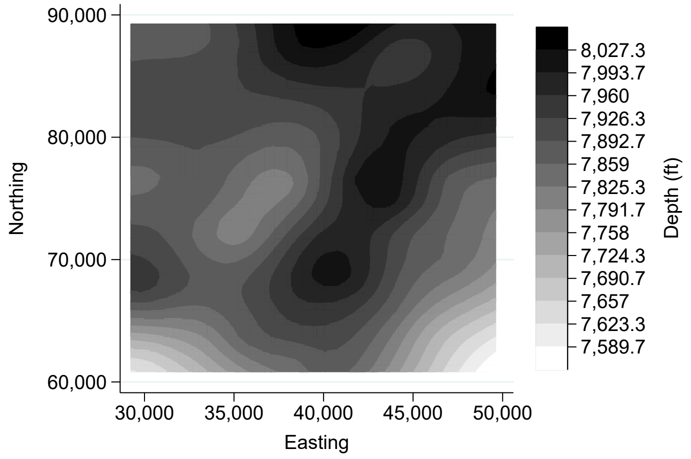
```
twoway contour depth northing easting, levels(15)
crule(linear) scolor(red) ecolor(blue)
```

This example is similar to the previous example except that the `ecolor(blue)` option specifies that blue be the ending color. The smallest depths are red, the highest depths are blue, and the intermediate depths are mixtures of red and blue, including greater amounts of blue with increasing depth. *Uses sandstone.dta*



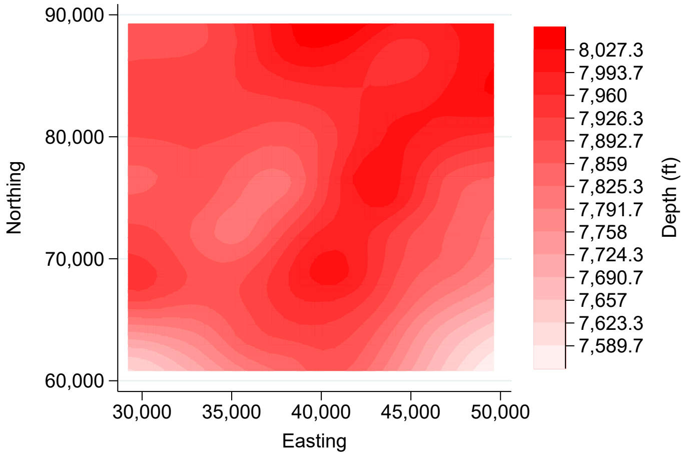
```
twoway contour depth northing easting, levels(15)
crule(linear) scolor(white) ecolor(black)
```

It might be simpler to use black and white in such graphs. In this example, the starting color is white and the ending color is black. The smallest value of depth is white, and the highest depth is black; darker colors correspond to greater values of `depth`. *Uses sandstone.dta*



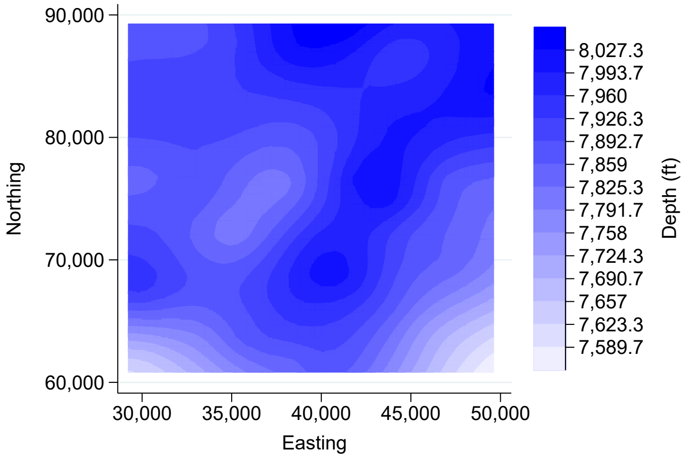
```
twoway contour depth northing easting, levels(15)
crule(intensity)
```

Rather than shifting from a starting and ending color, this example uses the `crule(intensity)` option. The result is that the contours are indicated by greater intensity of a single hue. The smallest levels of `depth` are displayed in very low intensity (nearly white). As depth increases, the intensity increases to light pink, then darker pink, and the greatest `depth` is shown in red. *Uses sandstone.dta*



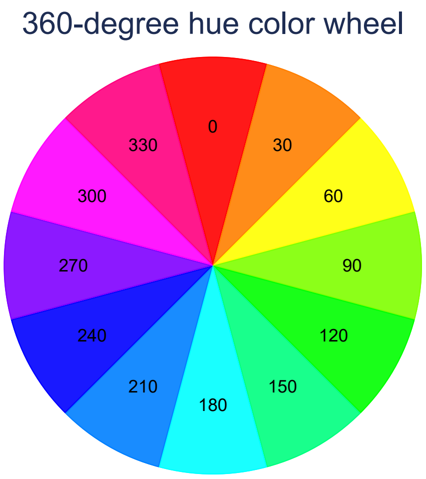
```
twoway contour depth northing easting, levels(15)
crule(intensity) ecolor(blue)
```

The `ecolor()` option can be used with the `crule(intensity)` option to specify the color used to represent the contours. In this example, contours are represented with different intensities of blue. Specifying the `scolor()` option has no effect when using the `crule(intensity)` option. *Uses sandstone.dta*

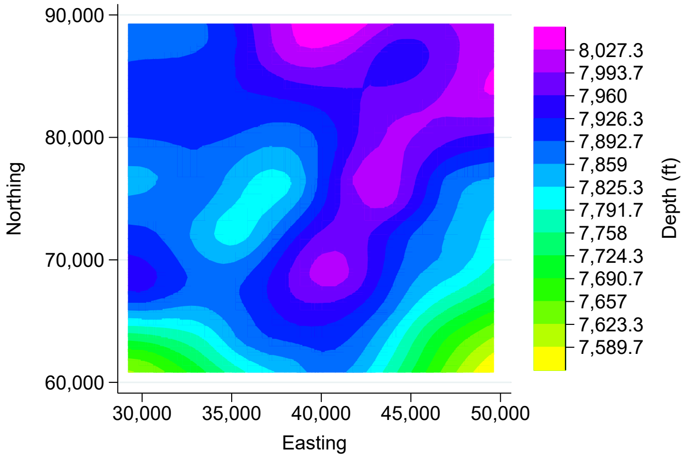


```
no command
```

A third method for indicating the colors of contours involves specifying a starting and ending color, and the intermediate colors are selected as the hues between the starting and ending colors. This pie chart illustrates a 360-degree color wheel that starts with red (at 0 degrees), then runs through orange, yellow, green, light blue, dark blue, and purple, and finally comes full circle back to red again. The following examples will refer back to this 360-degree color wheel.



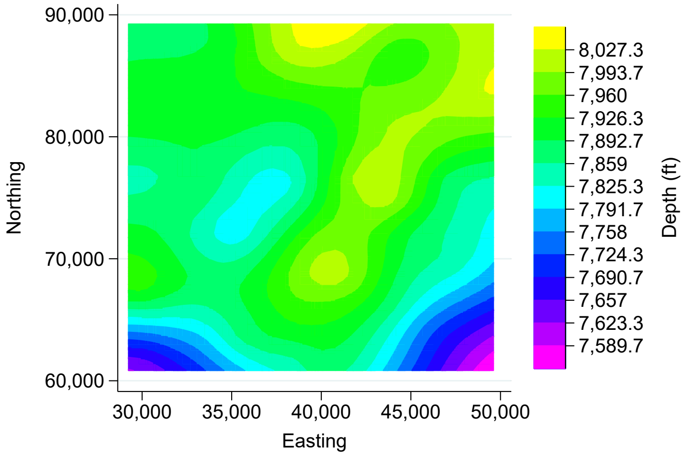
```
twoway contour depth northing easting, levels(15)
scolor(yellow) ecolor(purple) crule(hue)
```



In this example, the starting contour color is yellow and the ending contour is purple. Note how the intermediate values correspond to a clockwise rotation of hues from the color wheel, progressing from yellow to green, cyan, blue, and then ending at purple. We include the `crule(hue)` option, although not necessary, to emphasize that the intermediate values are chosen as a function of the hue. *Uses sandstone.dta*

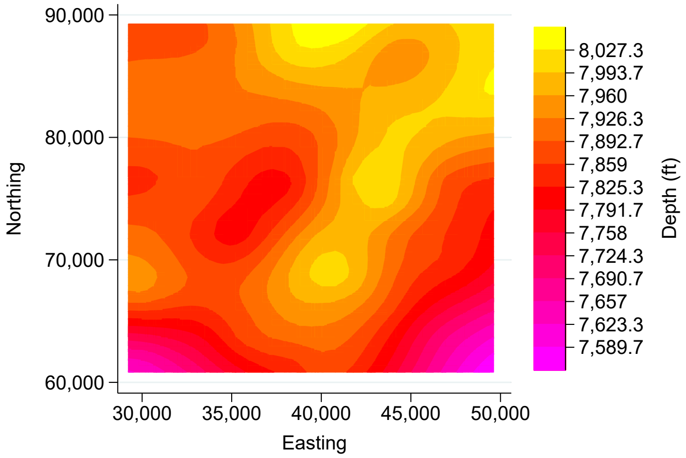
```
twoway contour depth northing easting, levels(15)
scolor(purple) ecolor(yellow) crule(hue)
```

In this example, the starting color is purple and the ending color is yellow. Because purple follows yellow on the color wheel (300 vs. 60), the hues in this example go in a counterclockwise direction based on the color wheel. As depth increases, the hues change from purple to blue, cyan, green, and then finally to yellow. *Uses sandstone.dta*



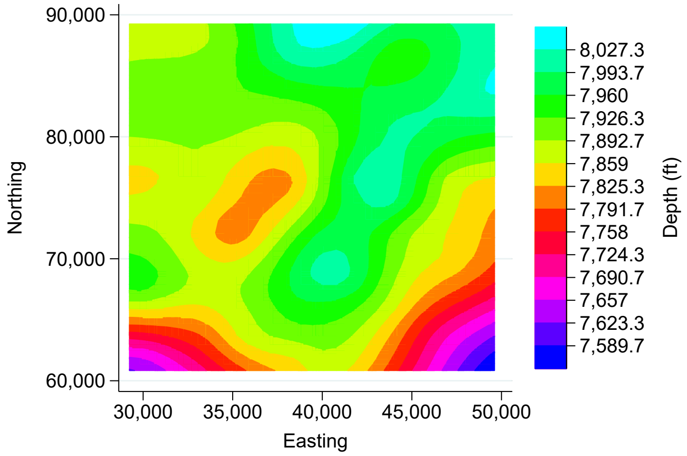
```
twoway contour depth northing easting, levels(15)
scolor(purple) ecolor(yellow) crule(chue)
```

This example uses the `crule(chue)` option. The result is that the contour colors go from purple to yellow in a clockwise fashion, showing purple, red, orange, and then yellow. The `crule(chue)` option works like the `crule(hue)` option if the ending color follows the starting color on the 360-degree color wheel. *Uses sandstone.dta*



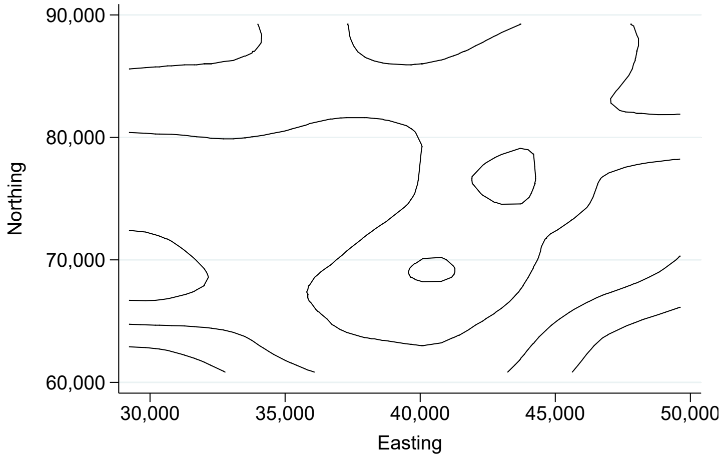
```
twoway contour depth northing easting, levels(15)
scolor(blue) ecolor(cyan) crule(chue)
```

This command is similar to the previous command except that the contour color starts with blue and ends with cyan. Starting with blue, the contour colors increase to purple, and then red, and then circle around to orange, yellow, green, and then end at cyan. *Uses sandstone.dta*



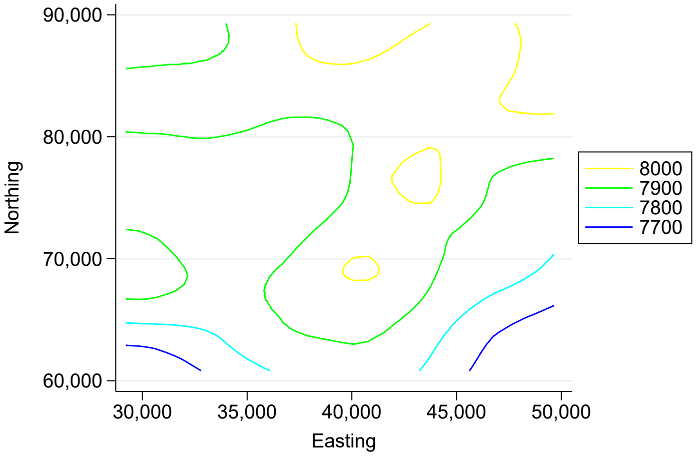
```
twoway contourline depth northing easting
```

The `twoway contourline` command creates contour plots except that the contours are shown (by default) as contour lines. *Uses sandstone.dta*



```
twoway contourline depth northing easting, colorlines
```

The `colorlines` option can be used to display the contour lines in color. Options previously illustrated with `twoway contour` (such as `ccolors()`, `scolor()`, and `ecolor()`) work similarly with `twoway contourline`. *Uses sandstone.dta*

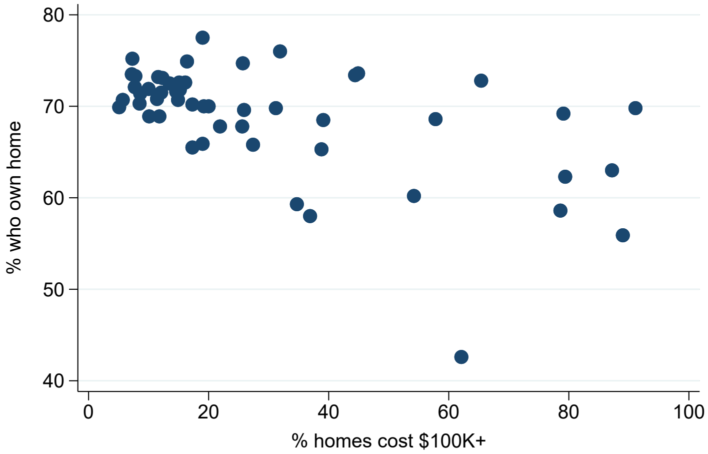

90,000
80,000
70,000
60,000
Northing
30,000
35,000
40,000
45,000
50,000
Easting
8000
7900
7800
7700

2.10 Options

This section discusses the use of options with `twoway`, showing the types of options we can use. For more details, see Options (section 8).

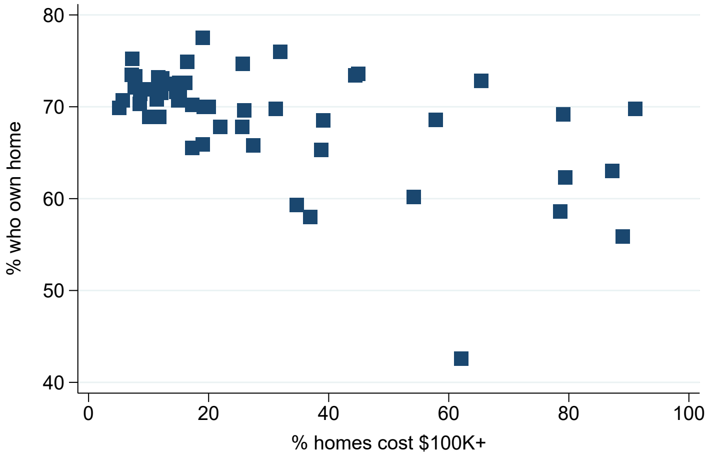
```
twoway scatter ownhome propval100
```

Consider this basic scatterplot. We will use this for illustrating options. *Uses allstates.dta*



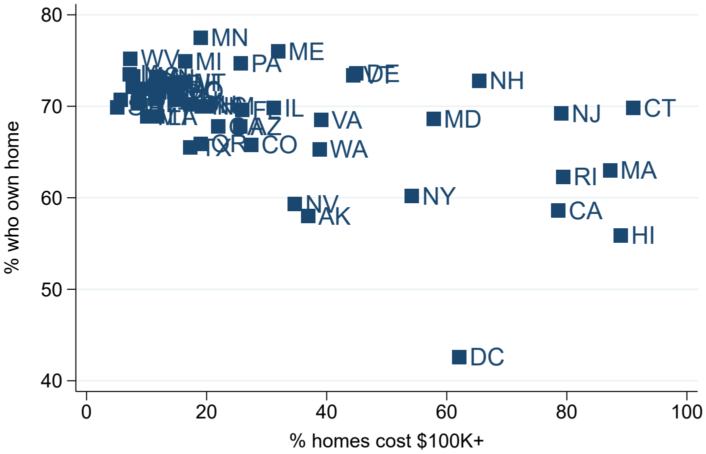
```
twoway scatter ownhome propval100, msymbol(S)
```

The `msymbol()` option controls the marker symbols. Here we use squares as symbols. See Options : Markers (section 8.1) for more details. *Uses allstates.dta*



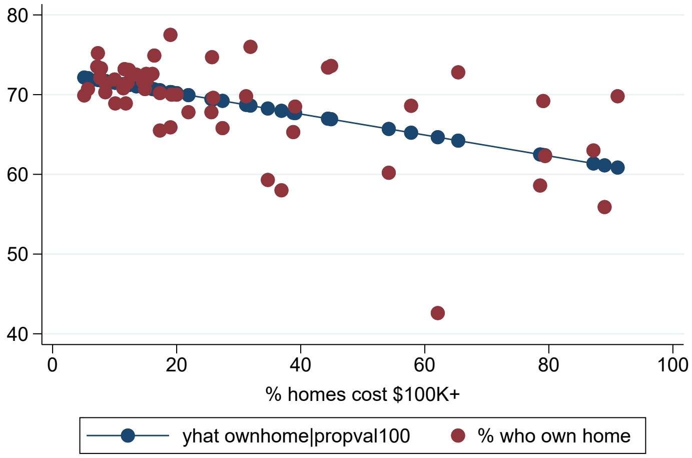
```
twoway scatter ownhome propval100, msymbol(S) mlabel(stateab)
```

The `mlabel()` option controls the marker labels. Here we label each marker with the variable `stateab`, showing the two-letter abbreviation for each state. See **Options : Marker labels** (section 8.2) for more information about marker labels. *Uses allstates.dta*



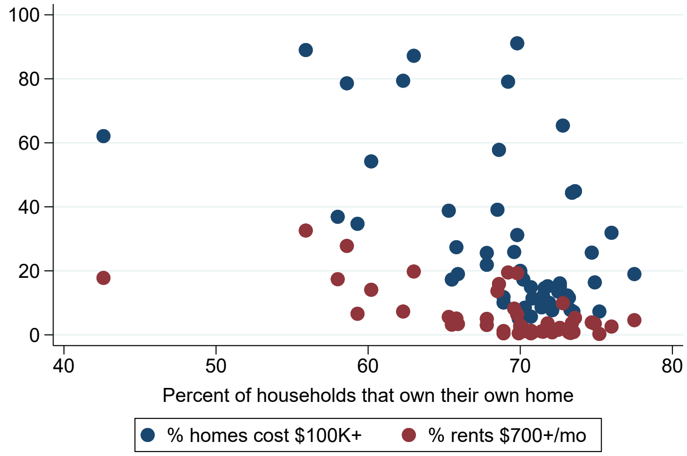
```
twoway scatter fv ownhome propval100, connect(l .) sort
```

Say that we regressed `ownhome` on `propval100` and generated predicted values named `fv`. Here we make a scatterplot and a fit line in the same graph using the `connect(l .)` option, which connects the values of `fv` but not the values of `ownhome`. We also add the `sort` option, which is generally recommended when using the `connect()` option. See **Options : Connecting** (section 8.3) for more details. *Uses allstates.dta*

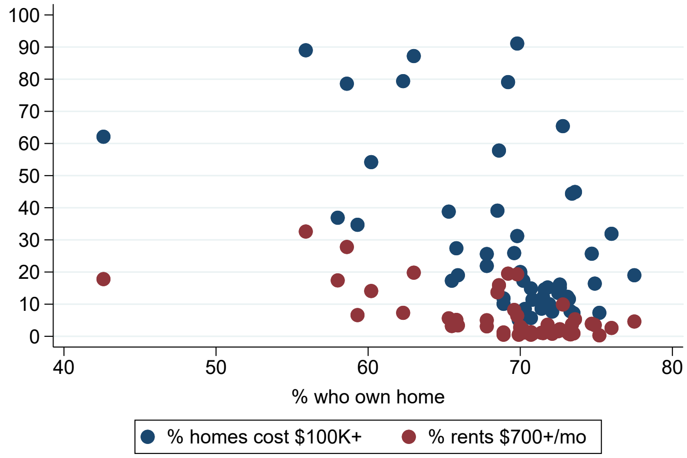


```
twoway scatter propval100 rent700 ownhome, xtitle(Percent of
households that own their own home)
```

We add a title to the x axis using the `xtitle()` option, as illustrated here. See **Options : Axis titles** (section 8.4) for more details about titles. *Uses allstates.dta*



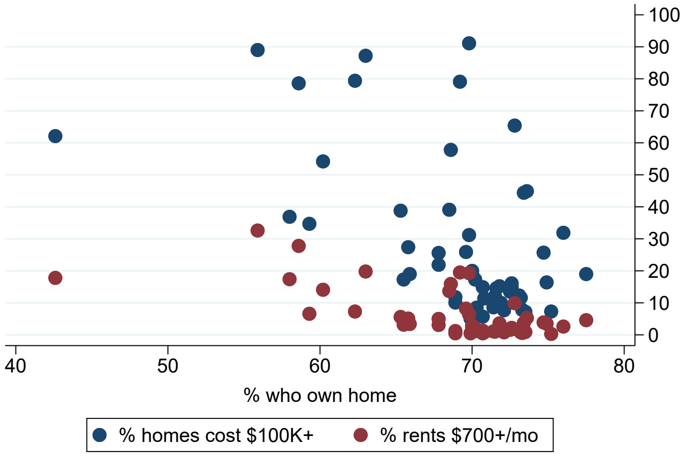
```
twoway scatter propval100 rent700 ownhome, ylabel(0(10)100)
```



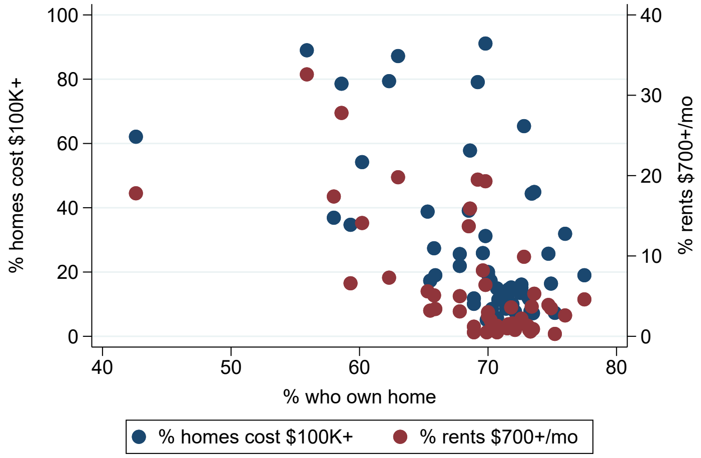
We can label the y axis from 0 to 100, incrementing by 10, using the `ylabel(0(10)100)` option, as shown here. See **Options : Axis labels** (section 8.5) for more information about labeling axes. *Uses allstates.dta*

```
twoway scatter propval100 rent700 ownhome, ylabel(0(10)100)
yscale(alt)
```

Stata provides several options you can use to control the axis scale for both the x and y axes. For example, here we use `yscale(alt)` to move the y axis to its alternate position, moving it from the left to the right. See **Options : Axis scales** (section 8.6) for more details about the options for controlling the axis scales. *Uses allstates.dta*



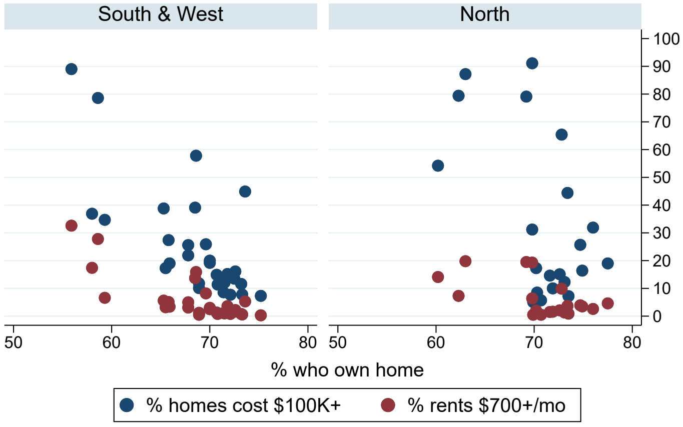
```
twoway (scatter propval100 ownhome) (scatter rent700 ownhome,
yaxis(2))
```



In this example, we show `propval100` by `ownhome` and also `rent700` by `ownhome`, but for this second plot, we put the y axis on the second y axis with the `yaxis(2)` option. See Options : Axis selection (section 8.7) for more information about using and controlling additional axes. *Uses allstates.dta*

```
twoway scatter propval100 rent700 ownhome, ylabel(0(10)100)
yscale(alt) by(north)
```

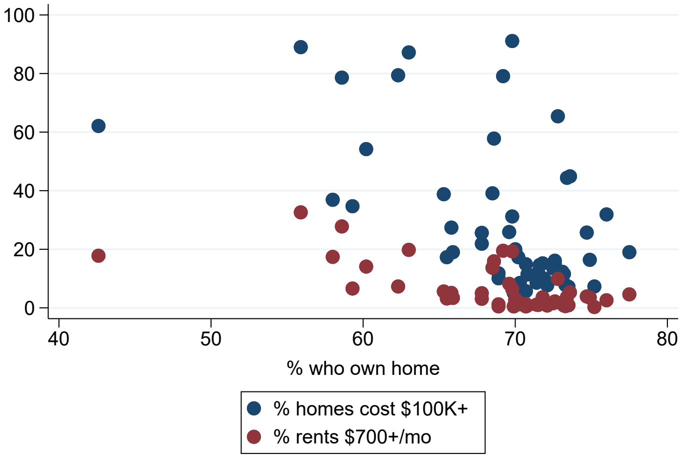
The `by()` option allows us to see a graph broken down by one or more `by()` variables. Here we show the previous graph further broken down by whether the state was part of the North, making two graphs that are combined into one graph. The section Options : By (section 8.8) shows more details and examples about the use of the `by()` option. *Uses allstates.dta*



Graphs by Region: North or not

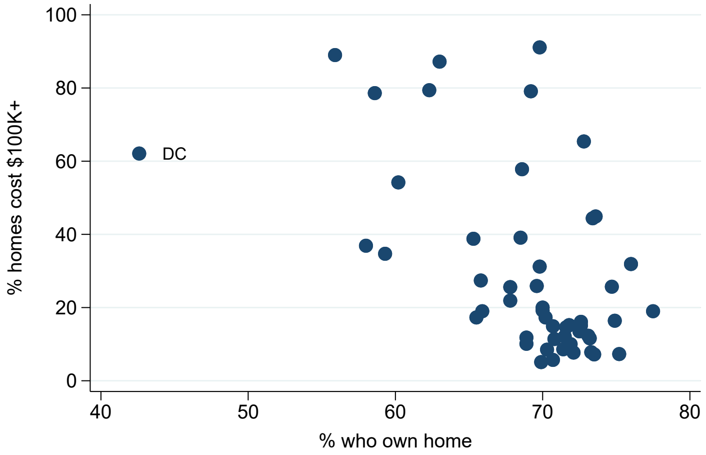
```
twoway scatter propval100 rent700 ownhome, legend(cols(1))
```

The `legend()` option allows us to control the contents and display of the legend. Here we use the `legend(cols(1))` option to indicate that we want the legend to display as one column. See Options : Legend (section 8.9) for more details about the `legend()` option. *Uses allstates.dta*



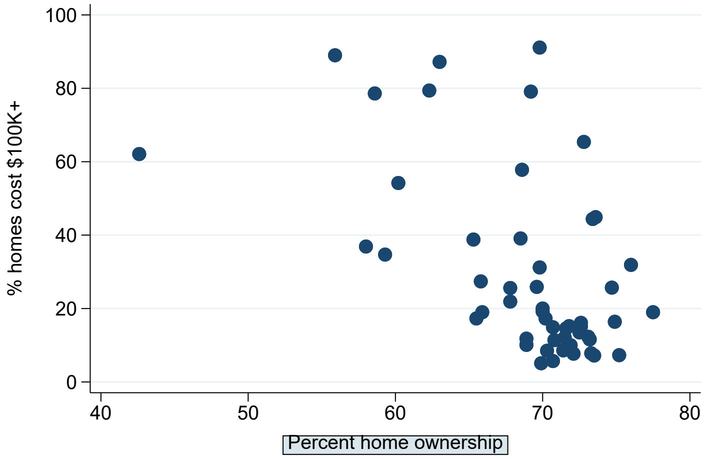
```
twoway scatter propval100 ownhome, text(62 45 "DC")
```

In this graph, there is one observation that stands out from the rest. Rather than use the `mlabel()` option to label all the markers, we may want to label just the outlying point. Thus we use the `text()` option to add the text `DC` at the (y,x) coordinates of (62,45), in effect labeling that point; see Options : Adding text (section 8.10) for more details. *Uses allstates.dta*



```
twoway scatter propval100 ownhome, xtitle(Percent home
ownership, box)
```

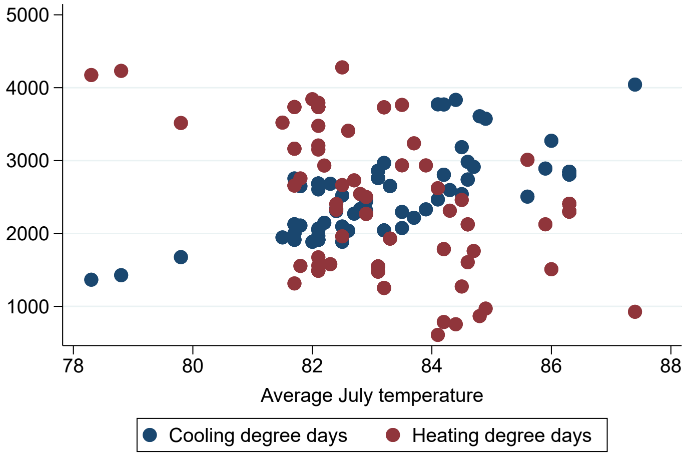
Most items of text on a Stata graph actually display within a box. We illustrate this with the `xtitle()` option, showing how we can place a box around this text. These options are described in more detail in Options : Textboxes (section 8.11). *Uses allstates.dta*



2.11 Overlaying plots

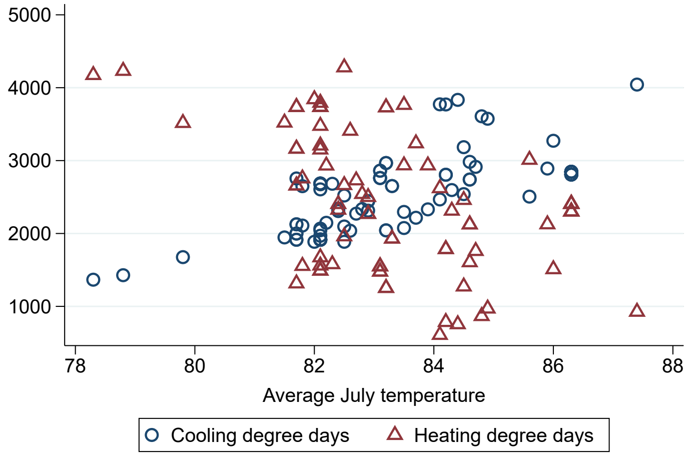
One of the features of `graph twoway` commands is the ability to overlay them to combine multiple plots into one graph. You can do this by graphing multiple y variables against one x variable in a `twoway` command. Or you can use multiple commands within a `twoway` command and overlay the resulting plots. You can also create separate graphs and glue them together using the `graph combine` command, which is discussed in Appendix : Save/Redisplay/Combine (section 11.4). The first set of examples shows how you can plot multiple y variables against one x variable using a `twoway` command. These examples use `citytemp.dta`, focusing just on the observations from division 7. This example includes observations that are so close that they could potentially conceal one another. This section will illustrate strategies to avoid such concealing, including adjusting opacity, where overlapping observations lead to overlapping of semitransparent colors. This topic is discussed in section 10.2.4.

```
graph twoway scatter cooldd heatdd tempjuly if division == 7
```



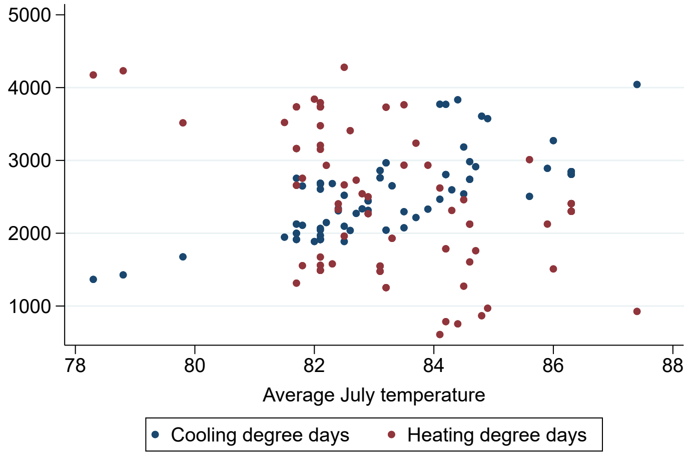
This graph shows `tempjuly` on the x axis and two variables on the y axis, `cooldd` (in blue) and `heatdd` (in red). When the graph is drawn, Stata first draws the markers for the first y variable (`cooldd` in blue) and then draws the markers for the second y variable (`heatdd` in red). Thus, some of the red markers are drawn on top of the blue markers, obscuring them. There is a miniclump of markers where $x = 82$ and $y = 2500$, and the red markers are obscuring a couple of blue markers. *Uses citytemp.dta*

```
twoway scatter cooldd heatdd tempjuly if division == 7,
msymbol(Oh Th)
```



Specifying hollow markers can help avoid the problem of markers being obscured. This example uses the `msymbol(Oh Th)` option to display `heatdd` using large hollow circles and `cooldd` using large hollow triangles. For example, note the appearance of the miniclump of markers where $x = 82$ and $y = 2500$—we can see a couple of blue circles underneath the red triangles. *Uses citytemp.dta*

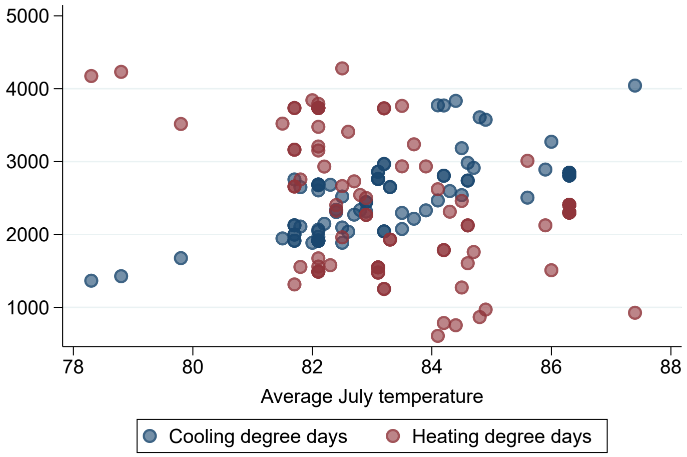
```
twoway scatter cooldd heatdd tempjuly if division == 7,
msize(small small)
```



Reducing the size of the markers can help avoid nearby markers from concealing one another. In this example, we use the option `msize(small small)`, which displays the markers for `heatdd` and `cooldd` both using small markers. Looking at the miniclump of markers where $x = 82$ and $y = 2500$, we can now see both the blue and red markers in that part of the graph. However, we might not want to make the scatterplot using such small markers. *Uses citytemp.dta*

```
twoway scatter cooldd heatdd tempjuly if division == 7,
mcolor(%60 %60)
```

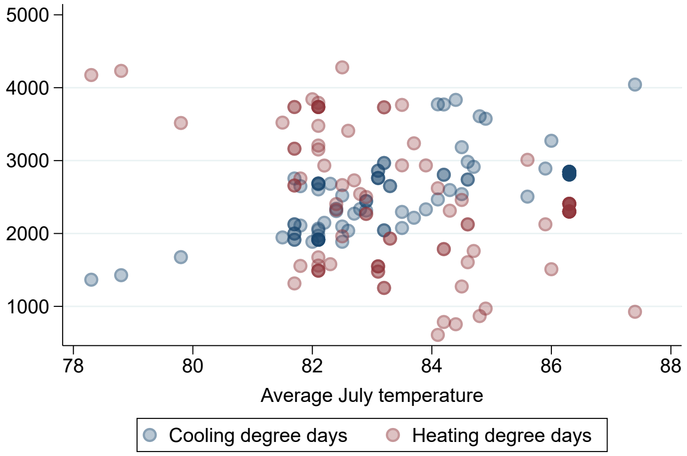
We can try reducing the opacity of the makers to see whether that helps to see both the red and blue markers where overlap occurs. In this example, we use the `mcolor(%60 %60)` option to make the opacity 60% for the marker for `heatdd` and likewise make the opacity 60% for the marker for `cooldd`.



Looking at the miniclump of markers where $x = 82$ and $y = 2500$, we can see both the blue and red markers, and the overlap of the markers shows as a mixture of red and blue. *Uses citytemp.dta*

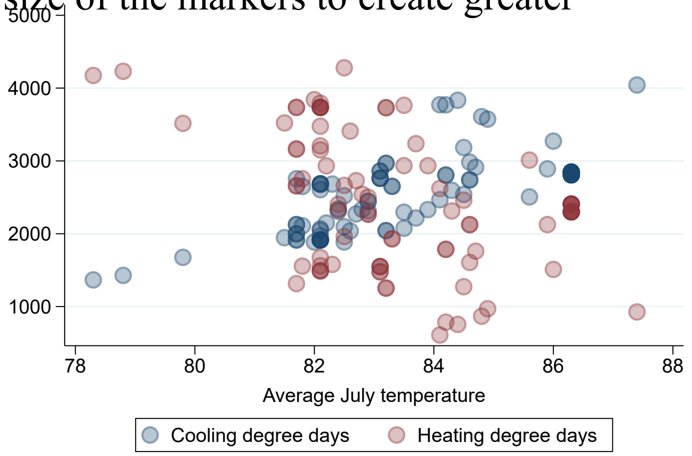
```
twoway scatter cooldd heatdd tempjuly if division == 7,
mcolor(%30 %30)
```

We can experiment with differing levels of opacity and see the impact on how the scatterplot looks. I like this version that uses even less opacity, specifying `mcolor(%30 %30)` to use 30% opacity for both `heatdd` and `cooldd`. *Uses citytemp.dta*



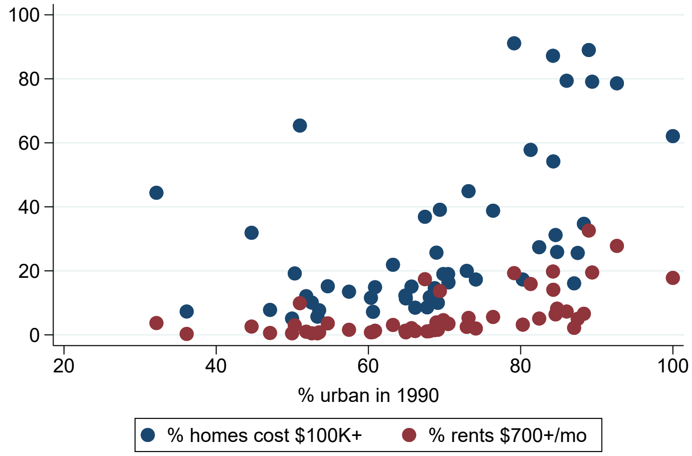
```
twoway scatter cooldd heatdd tempjuly if division == 7,
mcolor(%30 %30) msize(vlarge vlarge)
```

In this example, we increase the size of the markers to create greater overlap among markers. Because of the reduced opacity, overlapping markers combine in color. We can see instances of dark blue, where multiple cooling day observations overlap. We can see instances of dark red, where multiple heating day observations overlap. We can see purplish areas, where heating and cooling day observations overlap. *Uses citytemp.dta*



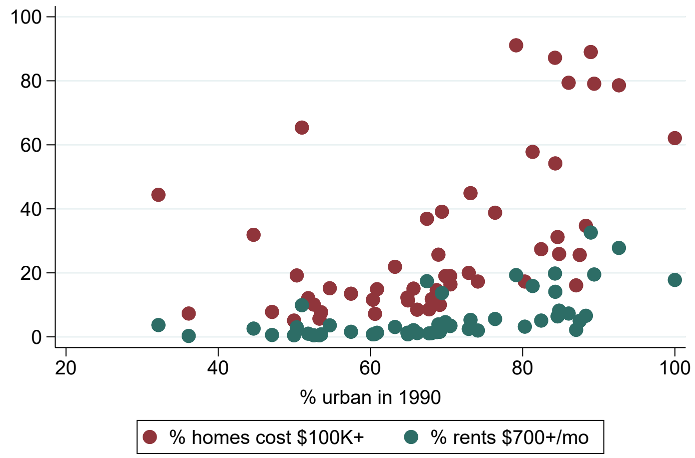
```
twoway scatter propval100 rent700 urban
```

The next set of examples use `allstates.dta`, graphing `propval100` and `rent700` on the y axis by `urban` on the x axis. *Uses allstates.dta*



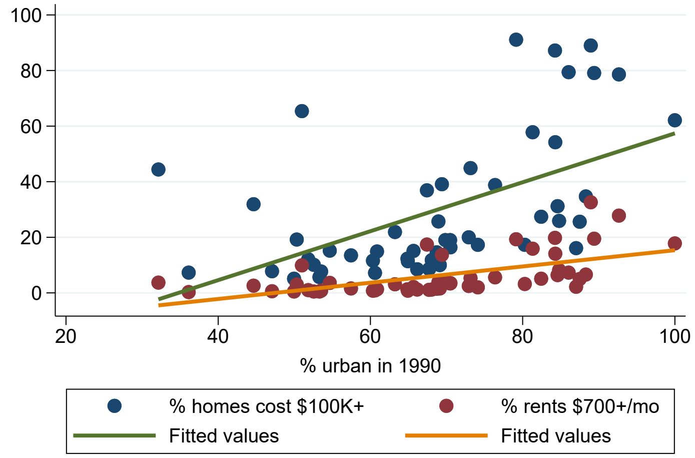
```
twoway scatter propval100 rent700 urban, mstyle(p2 p11)
```

We can use the `mstyle()` (marker style) option to choose among marker styles. These composite styles set the symbol, size, fill, color, outline color, and outline width for the markers. *Uses allstates.dta*



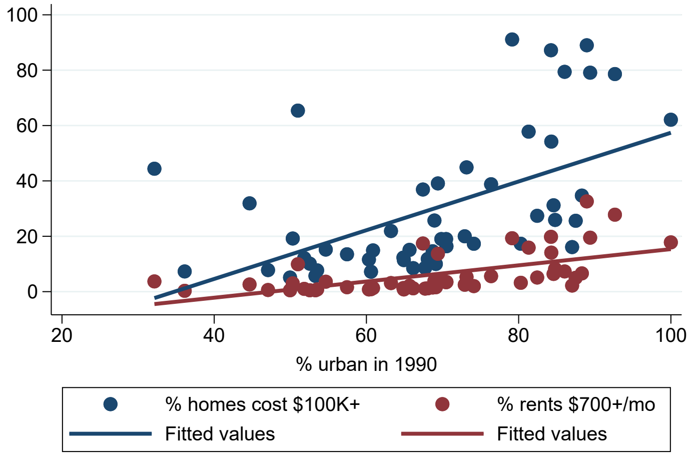
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban, lwidth(thick)) (lfit rent700 urban,
lwidth(thick))
```

Here we use `twoway scatter` twice to show a scatterplot of `propval100` and `rent700` against `urban`, and then we use `twoway lfit` twice to show the linear fit among these same variables. The colors of the scatterplot, however, do not match the fit lines. *Uses allstates.dta*



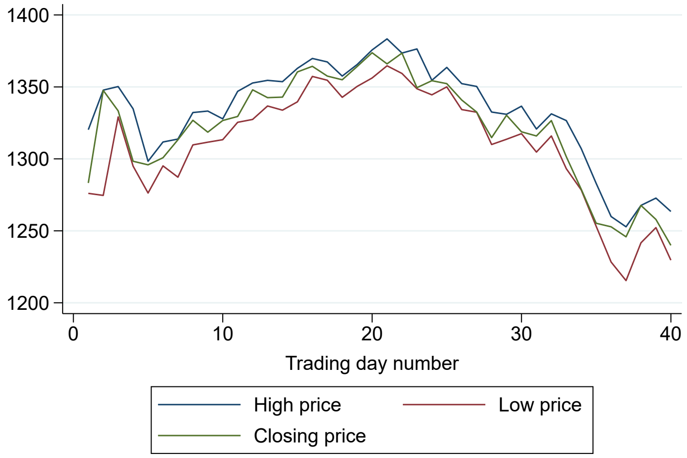
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban, lwidth(thick)) (lfit rent700 urban,
lwidth(thick)), pcycle(2)
```

By adding `pcycle(2)`, we indicate that the plot characteristics should repeat after two iterations. The first scatterplot is in blue, the second is in red, the third plot (the fit plot) switches back to red, and then the next fit plot is blue. This is a simple solution to make the scatterplot and fit lines match in color. *Uses allstates.dta*



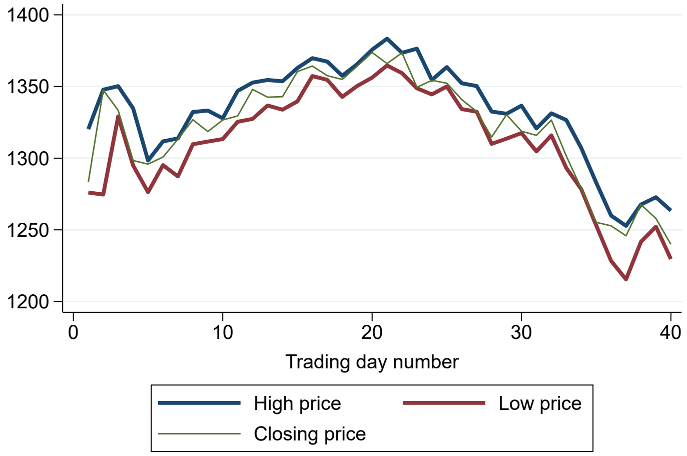
```
twoway line high low close tradeday, sort
```

We will briefly switch to using the `spjanfeb2001` data file. We can also graph multiple y variables against one x variable with a line graph. This works with `twoway line`, as illustrated here, as well as with `twoway connected` and `twoway tsline`. *Uses spjanfeb2001.dta*



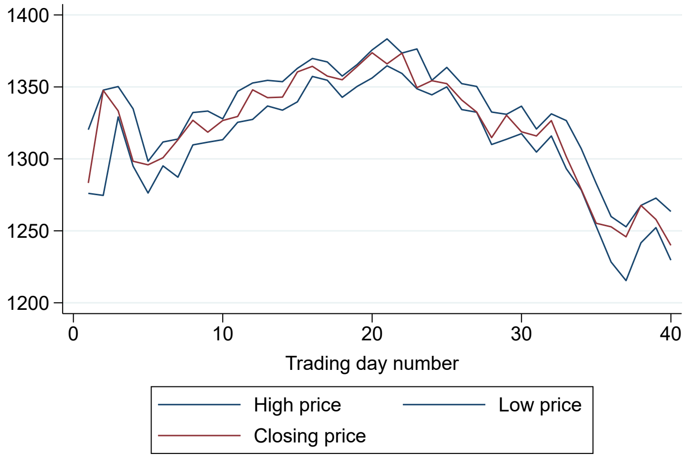
```
twoway line high low close tradeday, sort lwidth(thick thick
.)
```

Here we use the `lwidth()` option to change the width of the lines, making the lines for the high and low prices thick and leaving the line for the closing price at the default width. See **Styles : Linewidth** (section 10.7) for additional details about specifying line widths. *Uses spjanfeb2001.dta*



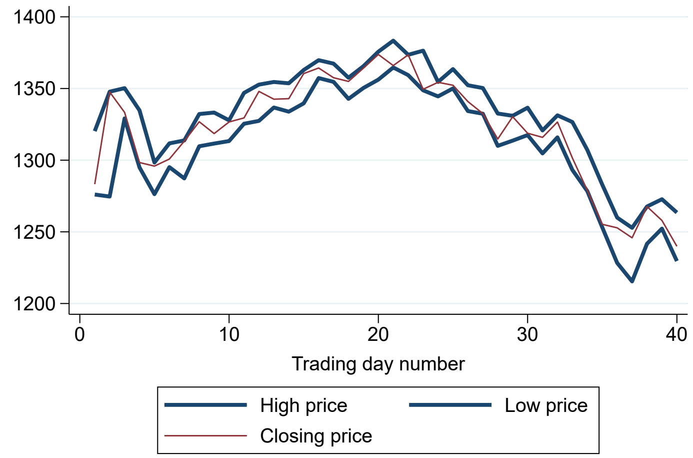
```
twoway line high low close tradeday, sort lstyle(p1 p1 p2)
```

When we graph multiple y variables, we can use `lstyle()` (line style) to control many characteristics of the lines at once. Here we plot the high and low prices with the same style, `p1`, and the closing price with a second style, `p2`. *Uses spjanfeb2001.dta*



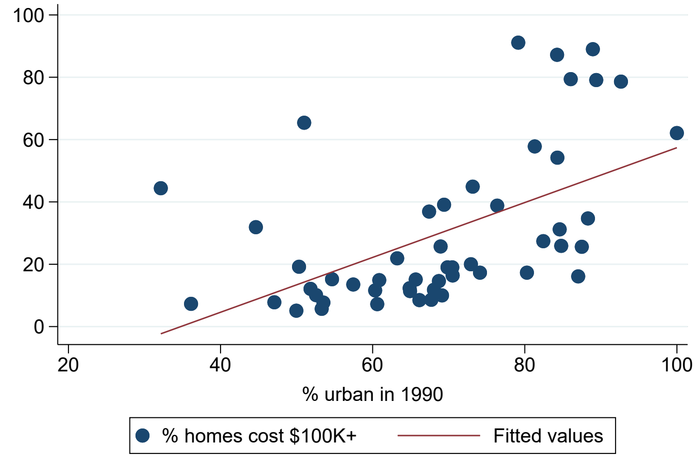
```
twoway line high low close tradeday, sort lstyle(p1 p1 p2)
lwidth(thick thick .)
```

In this example, we combine `lstyle()` and `lwidth()` to make the lines for the high and low prices the same style and both thick. The third line is drawn with the `p2` style, with the thickness left at its default value. *Uses spjanfeb2001.dta*



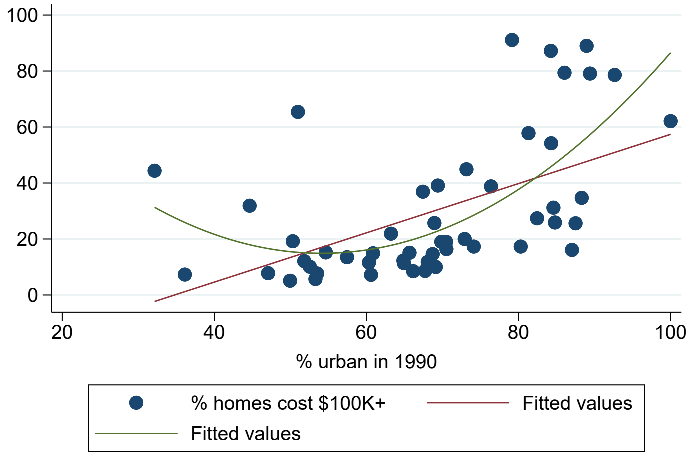
```
twoway (scatter propval100 urban) (lfit propval100 urban)
```

We return to the `allstates` data file. We can overlay multiple twoway graphs. Here we show a common kind of overlay: scatterplot overlaid with a linear fit between the two variables. Both the `scatter` command and the `lfit` command are surrounded by parentheses. *Uses allstates.dta*

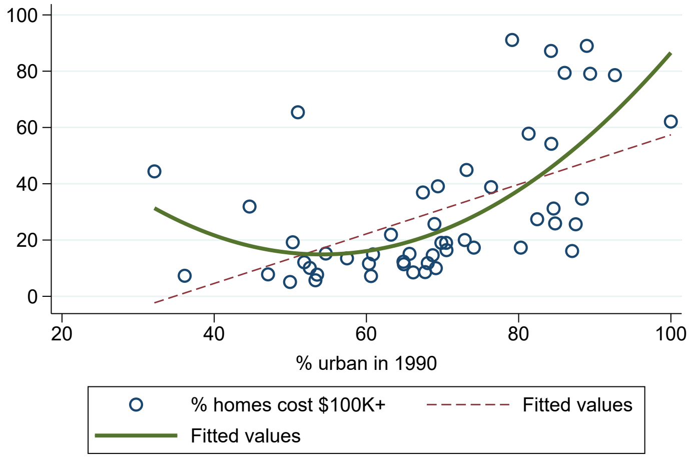


```
twoway (scatter propval100 urban) (lfit propval100 urban)
(qfit propval100 urban)
```

We can add a quadratic fit to the previous graph by adding a `qfit` command, so we can compare a linear fit and quadratic fit to see if there are nonlinearities in the fit. The legend does not clearly differentiate between the linear and quadratic fit; we will show how the legend can be modified to label this more clearly below. *Uses allstates.dta*



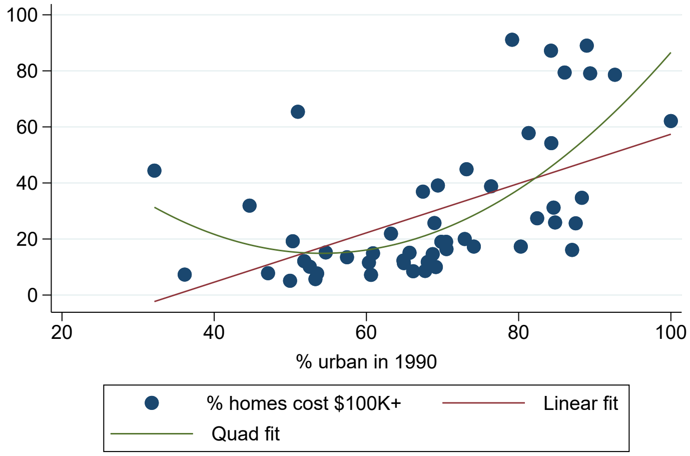
```
twoway (scatter propval100 urban, msymbol(Oh)) (lfit
propval100 urban, lpattern(dash)) (qfit propval100 urban,
lwidth(thick))
```



We add the `msymbol(Oh)` option to the `scatter` command, placing it after the comma, as it normally would be placed, but before the closing parenthesis, which indicates the end of the `scatter` command. We also add the `lpattern(dash)` option to the `lfit` command to make the line dashed and add the `lwidth(thick)` option to the `qfit` command to make the line thick. *Uses allstates.dta*

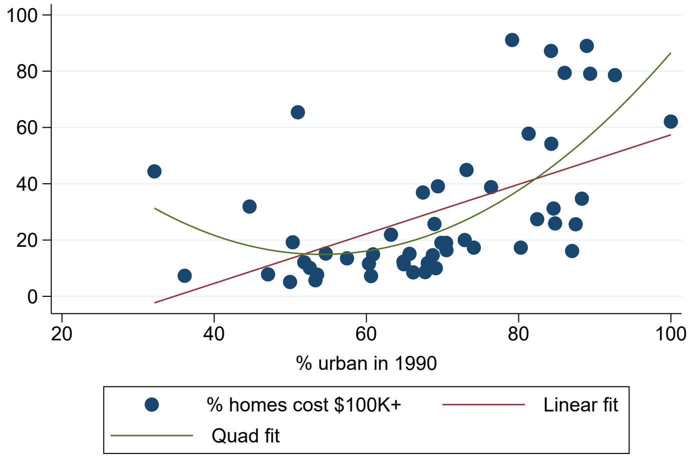
```
twoway (scatter propval100 urban) (lfit propval100 urban)
(qfit propval100 urban), legend(label(2 Linear fit) label(3
Quad fit))
```

Although each graph subcommand can have its own options, some options can apply to the entire graph. As illustrated here, we add a legend to the graph to clarify the difference in the fit values. This option appears following a comma after the closing parenthesis of the `qfit` command. The `legend()` option appears at the end of the command because it applies to the entire graph. *Uses allstates.dta*



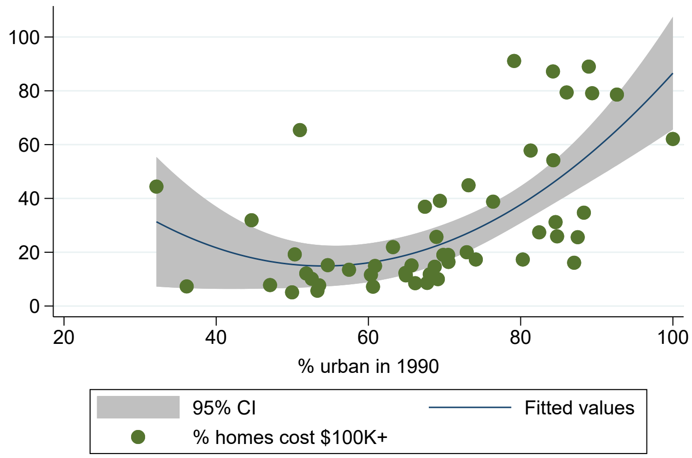
```
twoway (scatter propval100 urban) (lfit propval100 urban)
(qfit propval100 urban, legend(label(2 Linear fit) label(3
Quad fit)))
```

We can make the previous graph in a different, but less appropriate, way. The `legend()` option is given as an option of the `qfit()` command, not at the end as in the previous command. But Stata is forgiving of this, and even when such options are inappropriately given within a particular command, it treats them as though they were given at the end of the command. *Uses allstates.dta*



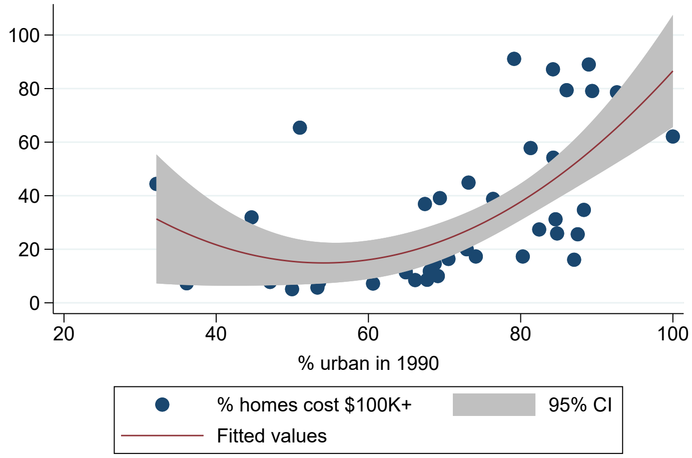
```
twoway (qfitci propval100 urban) (scatter propval100 urban)
```

Another common example of overlaying graphs is to overlay a fit line with a confidence interval and a scatterplot. *Uses allstates.dta*



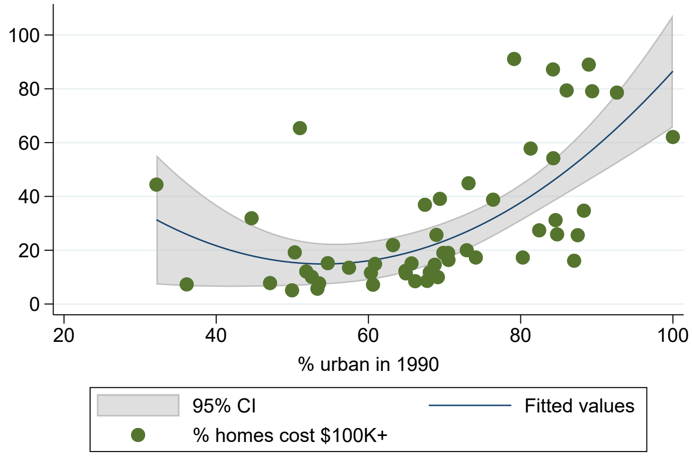
```
twoway (scatter propval100 urban) (qfitci propval100 urban)
```

However, note the order in which you overlay these two kinds of graphs. In this example, the `qfitci` was drawn after the `scatter`, and as a result, the points are obscured by the confidence interval. *Uses allstates.dta*



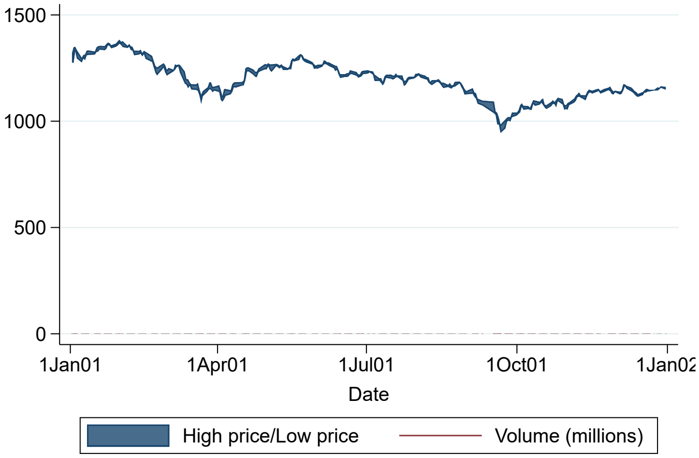
```
twoway (qfitci propval100 urban, fcolor(%50)) (scatter
propval100 urban)
```

Here we revert the order of the commands, specifying the `scatter` command last. I think this kind of graph looks better when the shaded area is shown with less opacity. We add the `fcolor(%50)` option to the `qfitci` command, and the fill color for the confidence interval is shown with 50% opacity. *Uses allstates.dta*



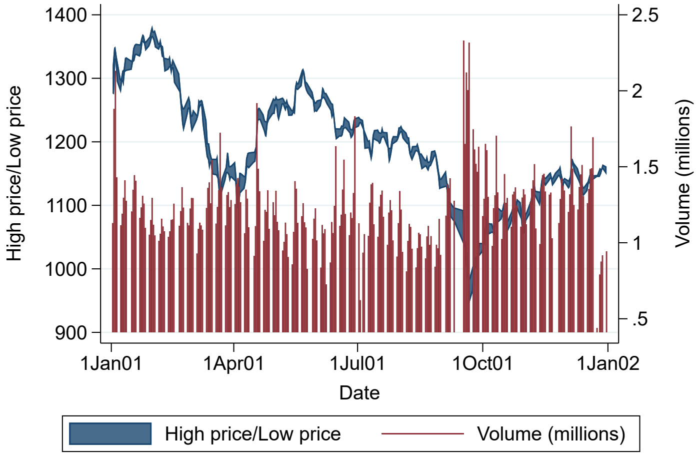
```
twoway (rarea high low date) (spike volmil date)
```

We now switch to the `sp2001ts` data file. Here we overlay the high and low closing prices with the volume of shares sold. However, because both are placed on the same y axis, it is difficult to see the spikes of `volmil` (volume in millions). *Uses sp2001ts.dta*



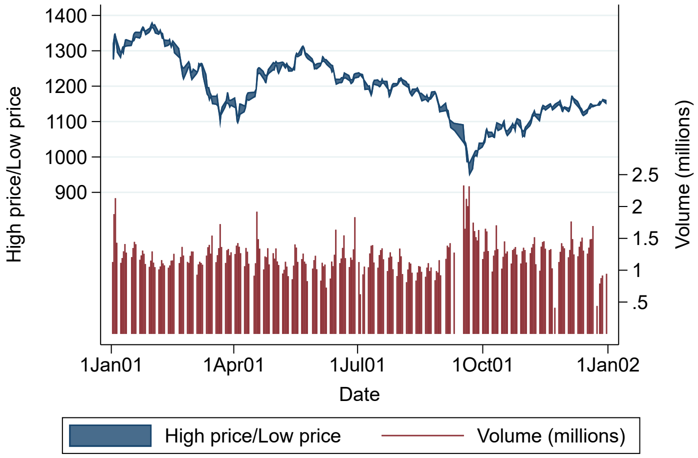
```
twoway (rarea high low date) (spike volmil date, yaxis(2)),
legend(span)
```

By placing `volmil` on the second y axis using the `yaxis(2)` option, we can now see the volume, but it obstructs the stock prices. We added the option `legend(span)` to allow the legend to be wider than the plot region of the graph. *Uses sp2001ts.dta*



```
twoway (rarea high low date) (spike volmil date, yaxis(2)),
legend(span) yscale(range(500 1400) axis(1)) yscale(range(0
5) axis(2))
```

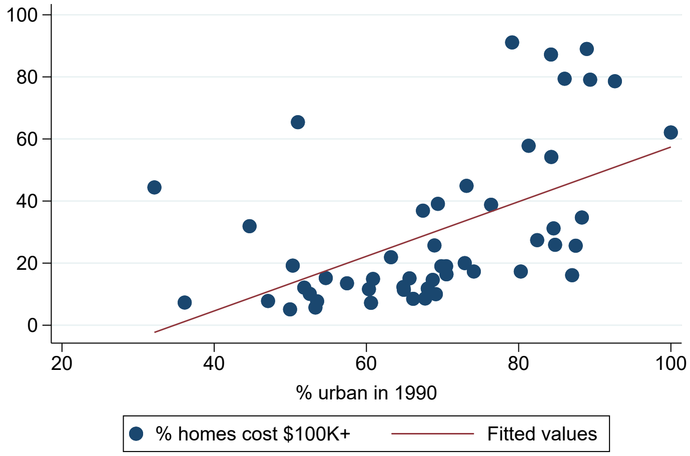
We use the `yscale()` option to modify the range for the first y axis to lift its range into the top third of the graph, and another `yscale()` option to modify the range for the second y axis, pushing the stock-market volume down to the bottom third. *Uses sp2001ts.dta*



Although the previous examples (and other examples in this book) have used the parenthetical notation for overlaid graphs, Stata also permits double vertical bars (`||`) for separating graphs. To illustrate this, let's repeat some of the graphs from above using this notation.

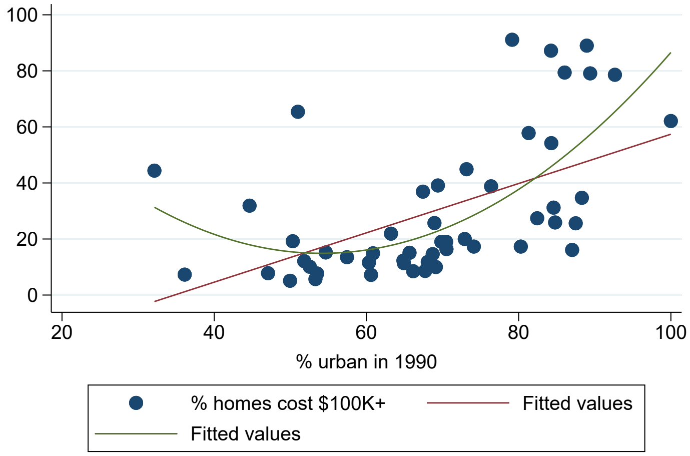
```
twoway scatter propval100 urban || lfit propval100 urban
```

We switch back to the `allstates` data file. Here the `||` notation is used to separate the `scatter` command from the `lfit` command. *Uses allstates.dta*



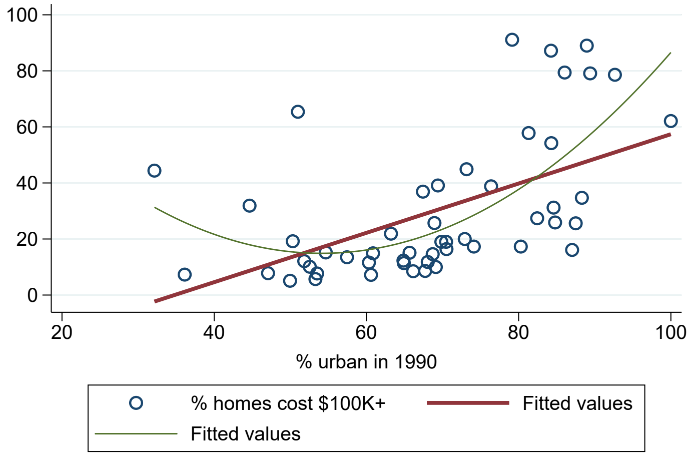
```
twoway scatter propval100 urban || lfit propval100 urban ||
qfit propval100 urban
```

Here we create three overlaid graphs by using the `||` notation. *Uses allstates.dta*



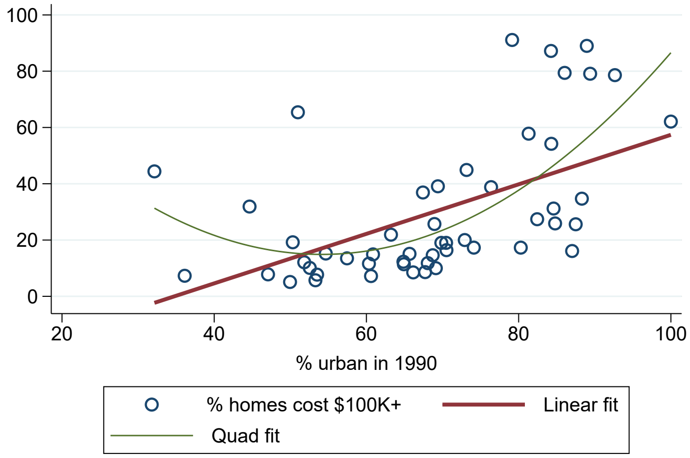
```
twoway scatter propval100 urban, msymbol(Oh) || lfit
propval100 urban, lwidth(thick) || qfit propval100 urban,
lwidth(medium)
```

This example shows how to use the `||` notation with options for each of the commands. *Uses allstates.dta*



```
twoway scatter propval100 urban, msymbol(Oh) || lfit
propval100 urban, lwidth(thick) || qfit propval100 urban,
lwidth(medium) ||, legend(label(2 Linear fit) label(3 Quad
fit))
```

This is another example using the `||` notation, in this case illustrating how to have options on each of the commands, along with `legend()` as an overall option. *Uses allstates.dta*



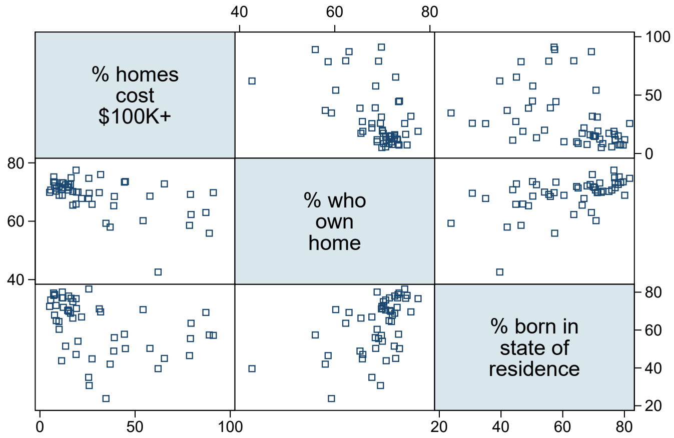
Chapter 3
Scatterplot matrix graphs

This chapter explores the use of the `graph matrix` command for creating scatterplot matrices among two or more variables. Many of the options that you can use with `graph twoway scatter` apply to these kinds of graphs as well; see Twoway : Scatter (section 2.1) and Options (section 8) for related information. This chapter illustrates the use of marker options and marker labels, as well as options for controlling the display of axes. It also includes options specific to the `graph matrix` command, as well as how to use the `by()` option. For more details about scatterplot matrices, see `help graph matrix`.

3.1 Marker options

This section looks at controlling and labeling the markers in scatterplot matrices. This section will show how to change the marker symbol, size, and color (both fill and outline color) and how to label the markers. You can label markers by using the `graph matrix` command just as you could when using the `graph twoway scatter` command. See also Options : Markers (section 8.1) and Options : Marker labels (section 8.2) for more details.

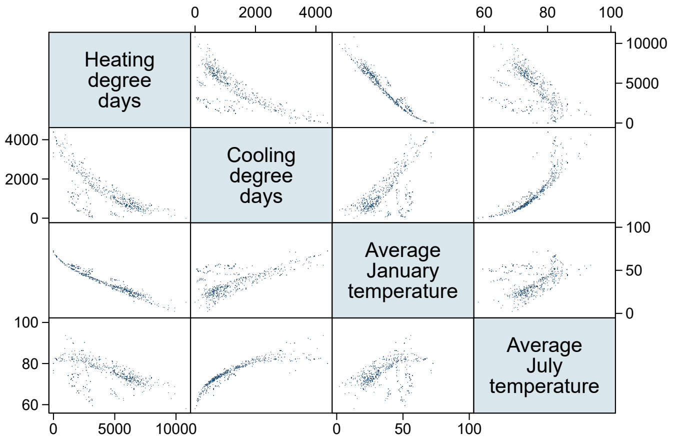
```
graph matrix propval100 ownhome borninstate, msymbol(Sh)
```



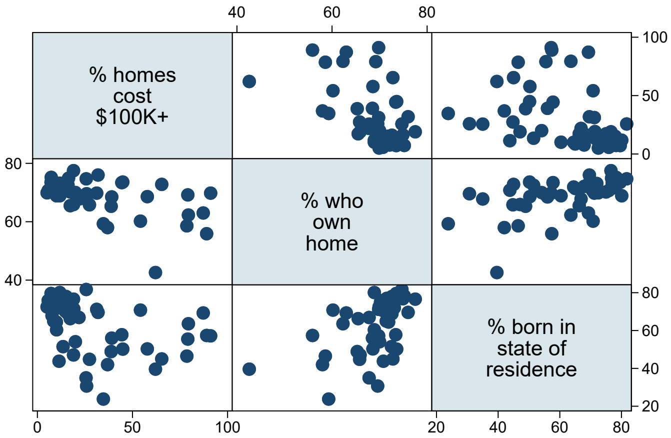
We can control the marker symbol with the `msymbol()` option. Here we make the symbols hollow squares. Values that we could specify include `D` (diamond), `O` (circle), `S` (square), `T` (triangle), and `X` (x). Using a lowercase letter (`d` instead of `D`) makes the symbol smaller. For diamonds, circles, squares, and triangles, you can append an `h` (for example, `Oh`) to indicate that the symbol should be hollow; see Styles : Symbols (section 10.11) for more examples. *Uses allstates.dta*

```
graph matrix heatdd cooldd tempjan tempjuly, msymbol(p)
```

When there are many observations, the `msymbol(p)` option can be useful because it displays a small point for each observation and can help us to see the overall relationships among the variables. Here we switch to the `citytemp` data file to illustrate this option. *Uses citytemp.dta*



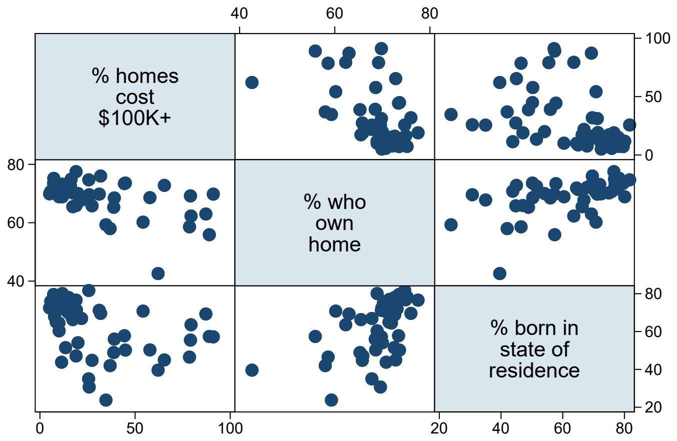
```
graph matrix propval100 ownhome borninstate, msize(vlarge)
```



Here we specify very large markers by using the `msize(vlarge)` (marker size). Other values we could have chosen include `vtiny`, `tiny`, `vsmall`, `small`, `medsmall`, `medium`, `medlarge`, `large`, `vlarge`, `huge`, `vhuge`, and `ehuge`; see Styles : Markersize (section 10.9) for more details (including how to specify sizes using points, inches, or centimeters). *Uses allstates.dta*

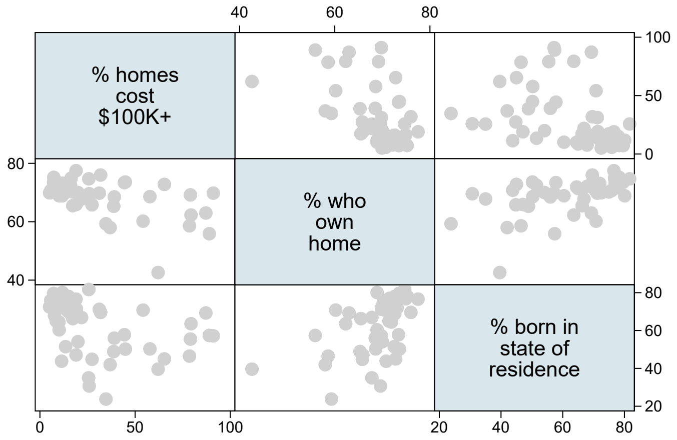
```
graph matrix propval100 ownhome borninstate, msize(*2)
```

If we prefer, we can specify the sizes as multiples of the marker's original size. Here we make the marker 2 times its original size by specifying `msize(*2)`. The asterisk is used to indicate that the value is # times the original. *Uses allstates.dta*



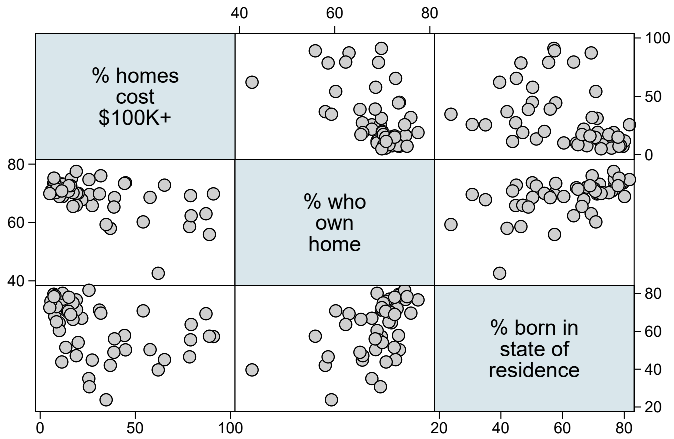
```
graph matrix propval100 ownhome borninstate, msize(vlarge)
mcolor(gs13)
```

Stata offers gray-scale colors named `gs0` (black) to `gs16` (white). Here we use the `mcolor()` (marker color) option to display markers that are very light gray; see Styles : Colors (section 10.2) for more information about specifying colors. *Uses allstates.dta*

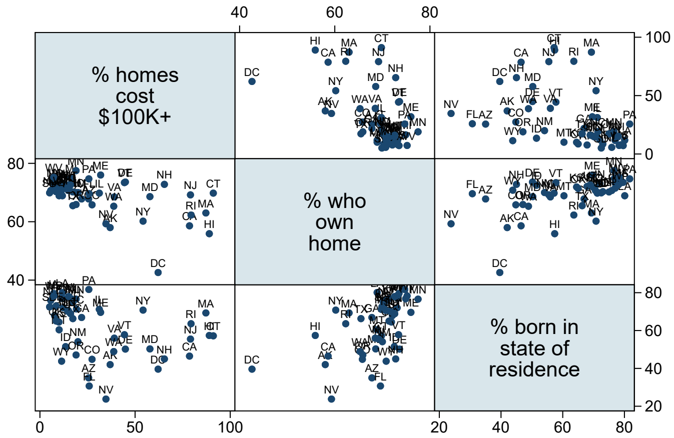


```
graph matrix propval100 ownhome borninstate, msize(vlarge)
mfcolor(gs13) mlcolor(black)
```

Here we use the `mfcolor()` (marker fill color) option to make the inside color of the marker `gs13` (light gray) and the `mlcolor()` (marker line color) option to make the outline color `black`. *Uses allstates.dta*



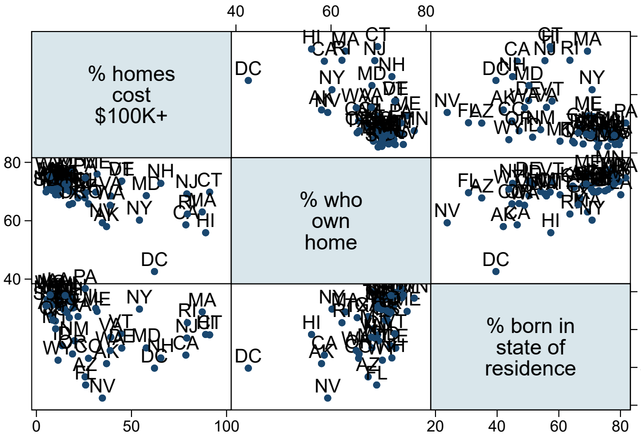
```
graph matrix propval100 ownhome borninstate, mlabel(stateab)
```



We can label the markers using the `mlabel()` (marker label) option. In this example, we label the markers with the two-letter state abbreviation by supplying the option `mlabel(stateab)`. Even though many of the labels overlap, the most interesting observations are those that stand out and have readable labels, such as DC and NV. For more details, see Options : Marker labels (section 8.2). *Uses allstates.dta*

```
graph matrix propval100 ownhome borninstate, mlabel(stateab)
mlabsize(large)
```

We can use the `mlabsize()` (marker label size) option to control the size of marker labels. Here we indicate that the marker labels should be large; see Options : Marker labels (section 8.2) for more details on controlling marker labels. *Uses allstates.dta*

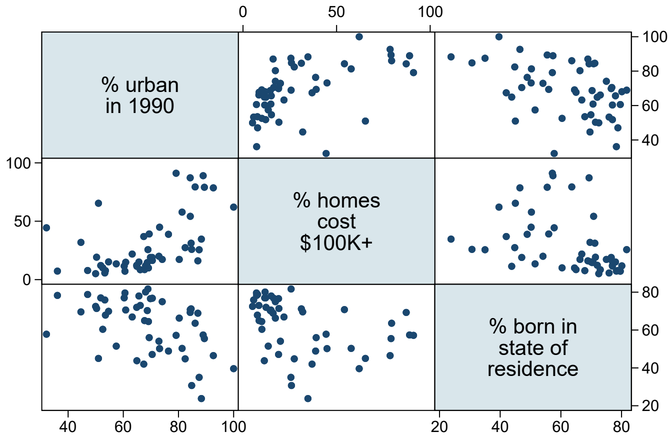

% homes cost $100K+
% who own home
% born in state of residence
40
60
80
100
50
0
80
60
40
80
60
40
20
0
50
100
20
40
60
80
DC
NY
NV
HI
CA
MA
CT
NJ
AK
FL
AZ

3.2 Controlling axes

This section looks at labeling axes in scatterplot matrices. It shows how to label axes of scatterplots, control the scale of axes, and insert titles along the diagonal. For more details, see Options : Axis labels (section 8.5), Options : Axis scales (section 8.6) and `help axis_options`.

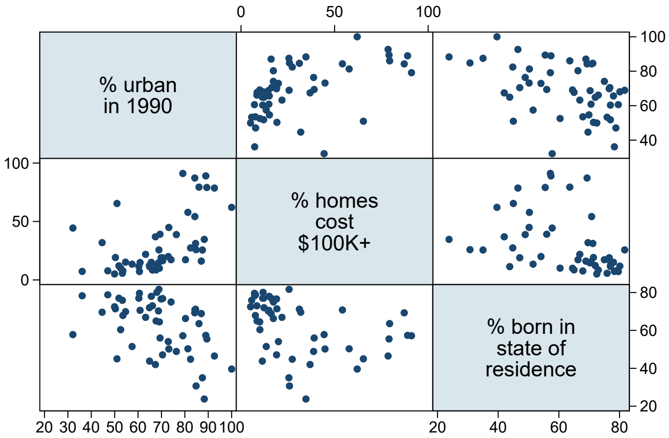
```
graph matrix urban propval100 borninstate
```

Let’s look at a scatterplot matrix of three variables: `urban`, `propval100`, and `borninstate`. *Uses allstates.dta*

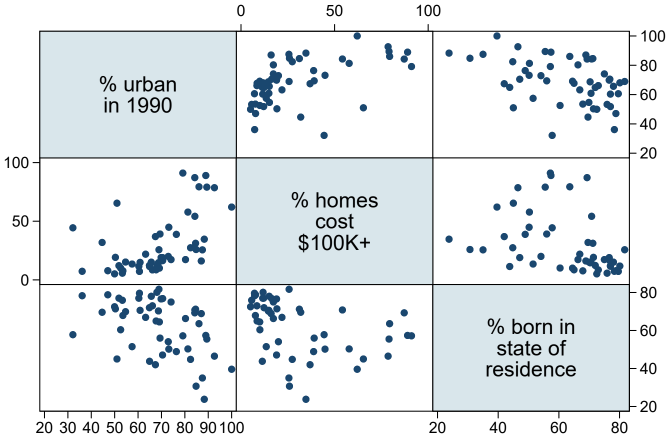


```
graph matrix urban propval100 borninstate, xlabel(20(10)100,
axis(1))
```

The way we control the axis labels with a scatterplot matrix is somewhat different from other kinds of graphs. Here we use the `xlabel()` option to label the first variable, `urban`, using labels going from 20 to 100 in increments of 10. This option applies to the first variable because we specified `axis(1)` within the `xlabel()` option. *Uses allstates.dta*



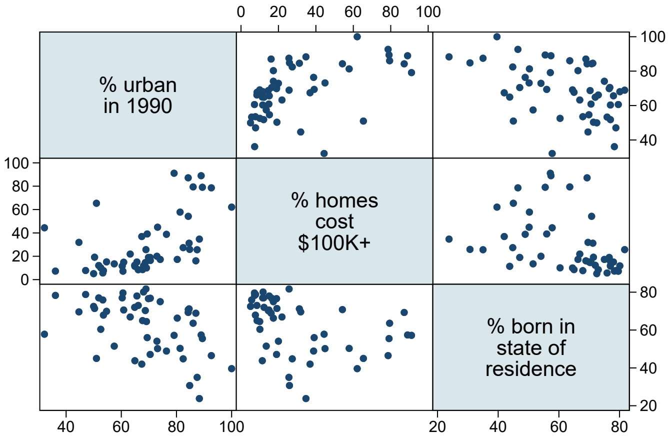
```
graph matrix urban propval100 borninstate, xlabel(20(10)100,
axis(1)) ylabel(20(20)100, axis(1))
```



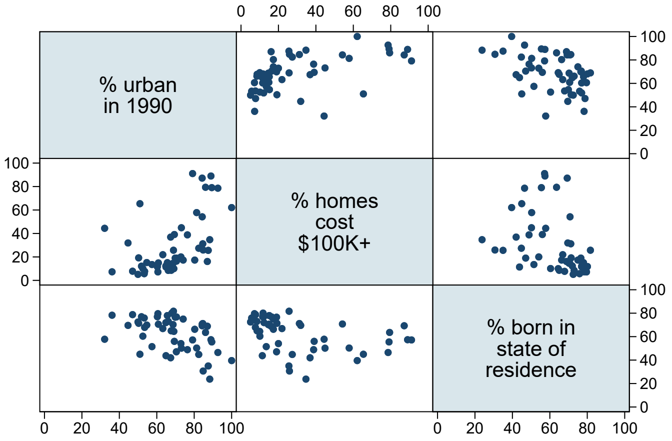
Here we alter the labels for the x axis and the y axis for the first variable, `urban`. The labels are changed on both axes because we specified `axis(1)` in both the `xlabel()` and the `ylabel()` options. We labeled the x axis from 20 to 100 incrementing by 10, and the y axis from 20 to 100 incrementing by 20. *Uses allstates.dta*

```
graph matrix urban propval100 borninstate, xlabel(0(20)100,
axis(2)) ylabel(0(20)100, axis(2))
```

We can change the label for the second variable, `propval100`, in a similar manner, by specifying `axis(2)`. Here we label the second variable ranging from 0 to 100 in increments of 20. *Uses allstates.dta*



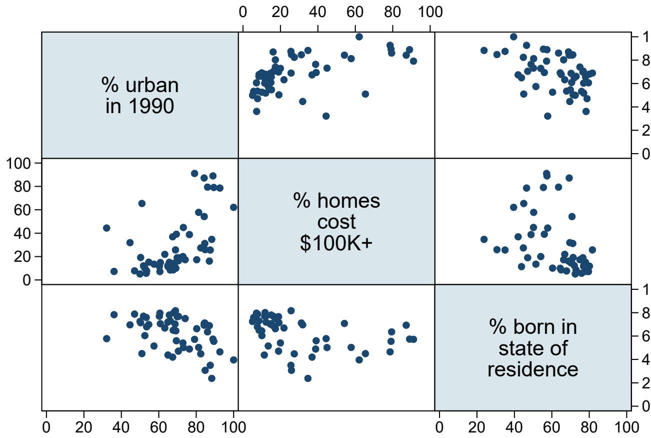
```
graph matrix urban propval100 borninstate, xlabel(0(20)100,
axis(1)) ylabel(0(20)100, axis(1)) xlabel(0(20)100, axis(2))
ylabel(0(20)100, axis(2)) xlabel(0(20)100, axis(3))
ylabel(0(20)100, axis(3))
```



Let’s label all these variables using the same scale, from 0 to 100 in increments of 20. As we can see, this involves quite a bit of typing, applying the `xlabel()` and `ylabel()` for `axis(1)`, `axis(2)`, and `axis(3)`, which applies this to the first, second, and third variables. However, the next example shows a more efficient way to do this. *Uses allstates.dta*

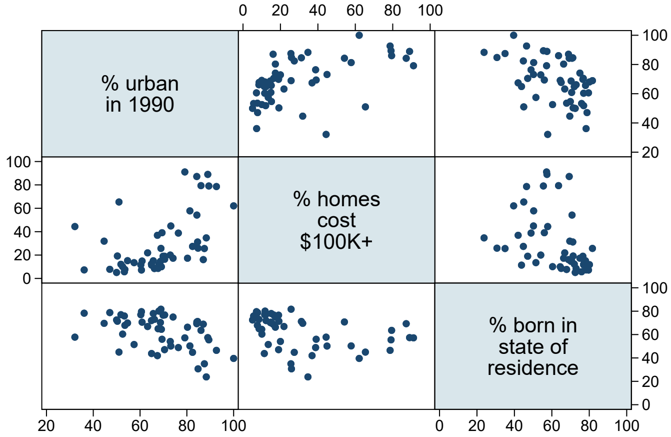
```
graph matrix urban propval100 borninstate,
maxes(xlabel(0(20)100) ylabel(0(20)100))
```

Stata has a simpler way of applying the same labels to all the variables in the scatterplot matrix by using the `maxes()` (multiple axes) option. This example labels the x and y axes from 0 to 100 with increments of 20 for all variables. *Uses allstates.dta*



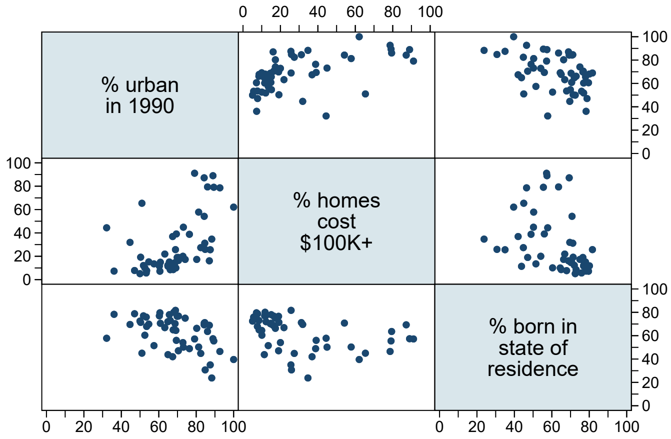
```
graph matrix urban propval100 borninstate,
maxes(xlabel(0(20)100) ylabel(0(20)100)) xlabel(20(20)100,
axis(1)) ylabel(20(20)100, axis(1))
```

We might want to label most of the variables in the scatterplot matrix the same way but with one or more exceptions in a different way. In this example, we label all the variables from 0 to 100, incrementing by 20, but then override the labeling for `urban` to make it 20 to 100, incrementing by 20. We do this by adding more `xlabel()` and `ylabel()` options that apply just to `axis(1)`. *Uses allstates.dta*



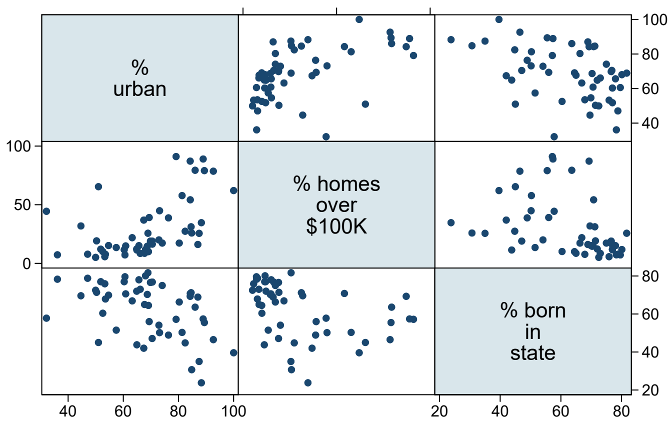
```
graph matrix urban propval100 borninstate,
maxes(xlabel(0(20)100) ylabel(0(20)100) xtick(0(10)100)
ytick(0(10)100))
```

Here we label all the variables from 0 to 100, incrementing by 20, and also add ticks from 0 to 100, incrementing by 10. The `xtick()` and `ytick()` options work the same way as the `xlabel()` and `ylabel()` options. We place these options within the `maxes()` option, and they apply to all the axes. See Options : Axis labels (section 8.5) and Options : Axis scales (section 8.6) for more details. *Uses allstates.dta*



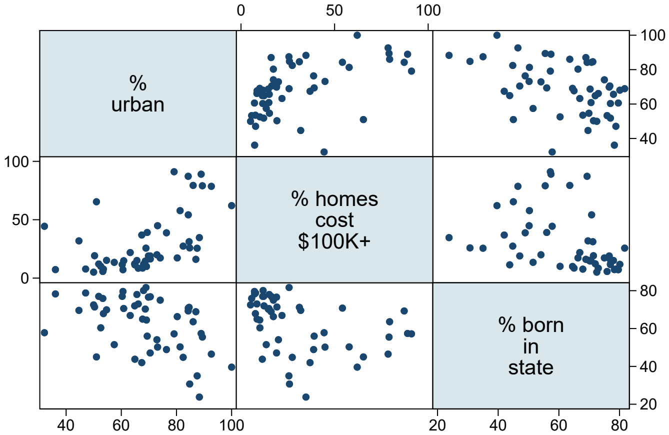
```
graph matrix urban propval100 borninstate, diagonal("% urban"
"% homes over $100K" "% born in state")
```

When we use `twoway scatter`, we control the titles by using the `xtitle()` and `ytitle()` options. When using `graph matrix`, we control the titles by using the `diagonal()` option. Here we use the `diagonal()` option to change the titles for all variables. *Uses allstates.dta*



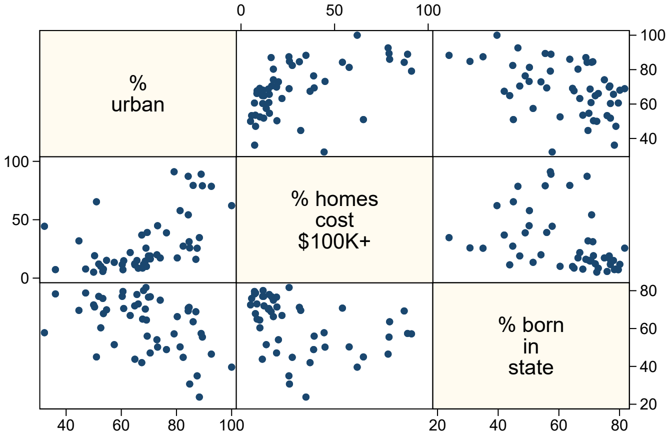
```
graph matrix urban propval100 borninstate, diagonal("% urban"
. "% born in state")
```

We do not have to change all the titles. If we want to change just some of the titles, we can place a period (`.`) for the labels that we want to stay the same. In this example, we change the titles for the first and third variables but leave the second as is. *Uses allstates.dta*



```
graph matrix urban propval100 borninstate, diagonal("% urban"
. "% born in state", fcolor(eggshell))
```

We can control the display of the titles by using textbox options. For example, we make the background color of the text area eggshell by using the `fcolor(eggshell)` option. See Options : Textboxes (section [8.11](#)) for more examples of textbox options. *Uses allstates.dta*

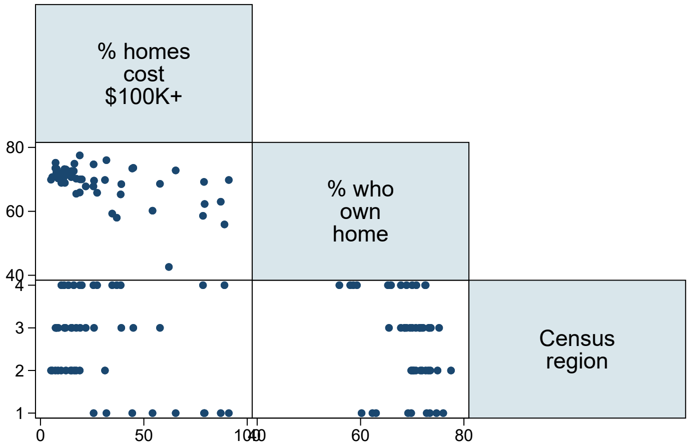


3.3 Matrix options

This section discusses options that we can use to control the look of the scatterplot matrix, including showing just the lower half of the matrix, jittering markers, and scaling the size of marker text. For more details, see `help graph matrix`.

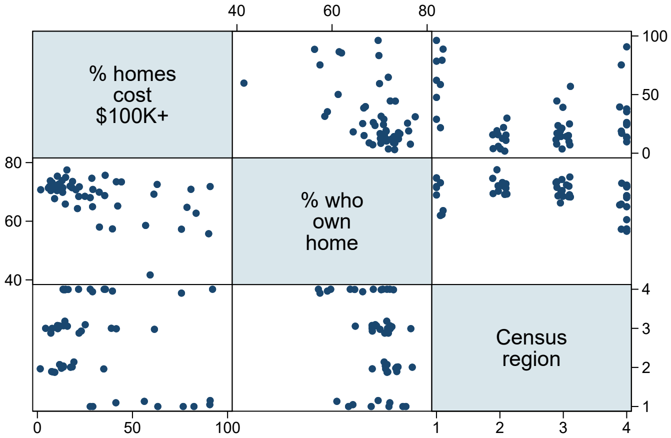
```
graph matrix propval100 ownhome region, half
```

The `half` option displays only the lower diagonal of the scatterplot matrix. *Uses allstates.dta*

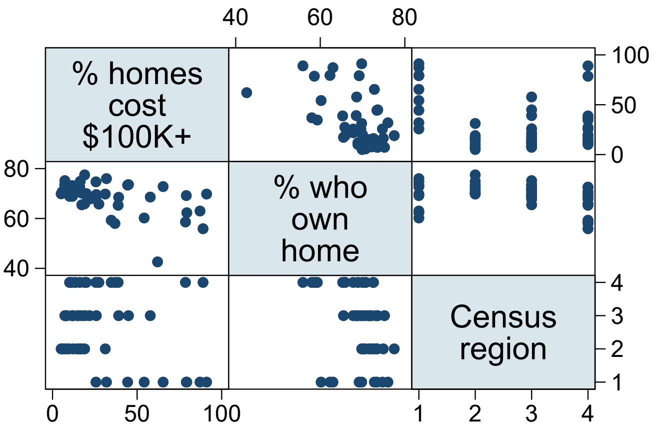


```
graph matrix propval100 ownhome region, jitter(3)
```

The `jitter()` option adds random noise to the points; the higher the value given, the more random noise is added. This option is especially useful when many observations have the same (x,y) values, so several observations can appear as one point. *Uses allstates.dta*



```
graph matrix propval100 ownhome region, scale(1.5)
```



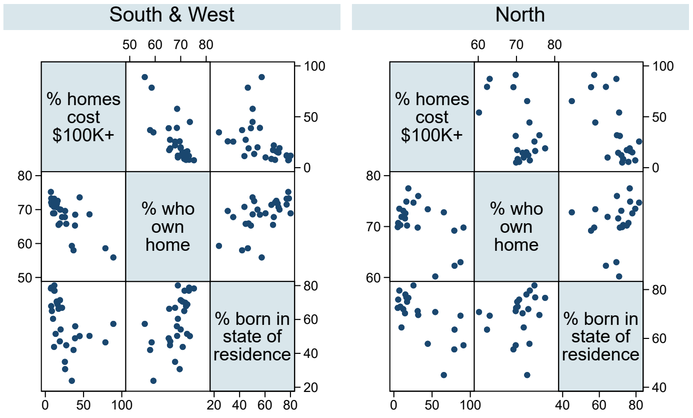
We can use the `scale()` option to magnify the contents of the graph, including the markers, labels, and lines, but not the overall size of the graph. Here we increase the size of these items, making them 1.5 times their normal size. Unlike other similar options, this option does not take an asterisk preceding the value; that is, we specify 1.5, not *1.5. *Uses allstates.dta*

3.4 Graphing by groups

This section looks at the use of the `by()` option for showing separate graphs based on the levels of a `by()` variable. For more information, see Options : By (section 8.8) and `help by_option`.

```
graph matrix propval100 ownhome borninstate, by(north)
```

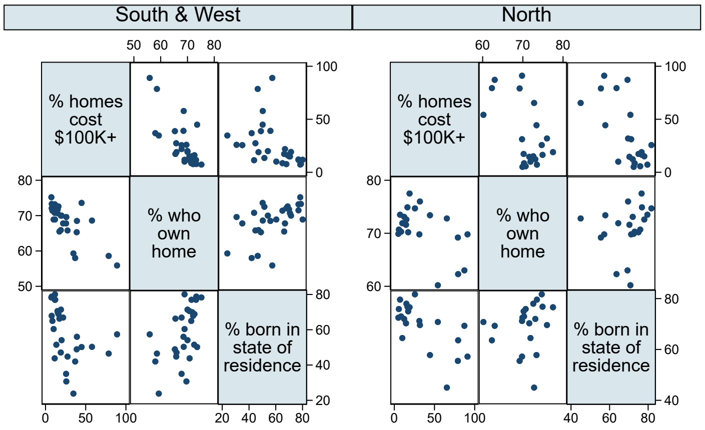
We can use the `by()` option with `graph matrix` to show separate scatterplot matrices by a particular variable. Here we show separate scatterplot matrices for households in northern states and nonnorthern states. *Uses allstates.dta*



Graphs by Region: North or not

```
graph matrix propval100 ownhome borninstate, by(north,
compact)
```

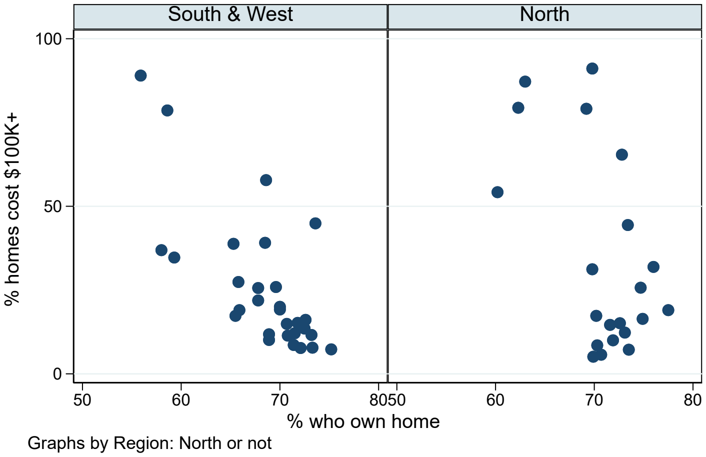
To display the graphs closer together, we use the `compact` option. *Uses allstates.dta*



Graphs by Region: North or not

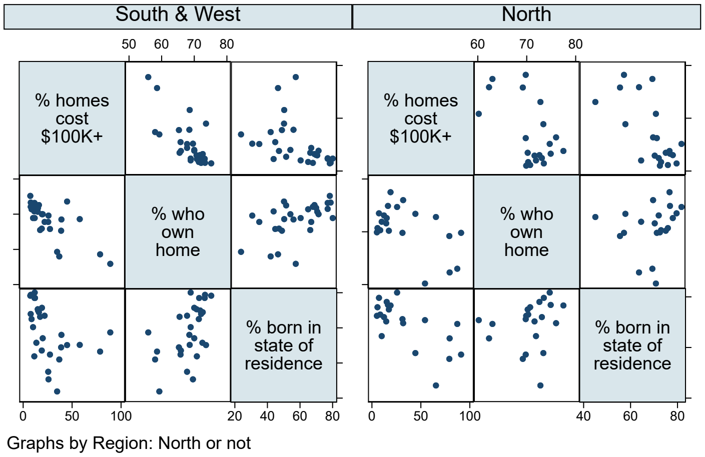
```
twoway scatter propval100 ownhome, by(north, compact)
```

If we compare the previous scatterplot matrix with this twoway scatterplot, we see that the `compact` option does not make the scatterplot matrix as compact as it does with a `twoway scatter` command, which joins the two graphs on their edges by omitting the y labels between the two graphs. *Uses allstates.dta*



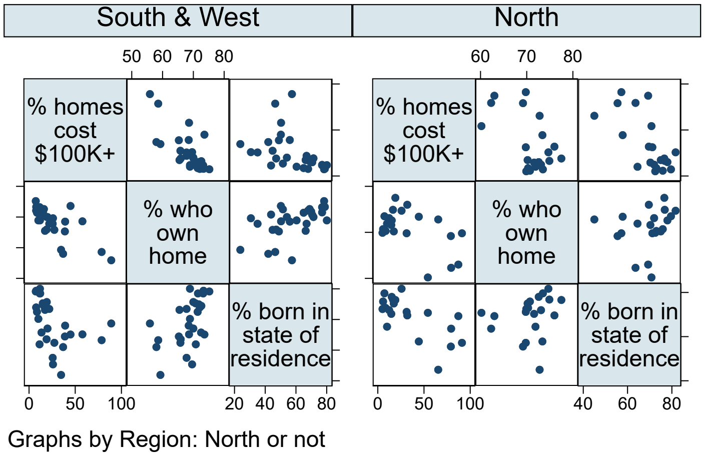
```
graph matrix propval100 ownhome borninstate, by(north,
compact) maxes(ylabel(, nolabels))
```

We can make the `graph matrix` display more compactly with the `by()` option by using the `maxes(ylabel(, nolabels))` option to suppress the labels on all the y axes. Then, when we use the `compact` option, the edges of the plots are pushed closer together. *Uses allstates.dta*



```
graph matrix propval100 ownhome borninstate, by(north,
compact scale(*1.3)) maxes(ylabel(, nolabels))
```

We use `scale()` within the `by()` option to increase the size of the markers, labels, and text to make them more readable. `scale()` is especially useful for the display of small graphs. *Uses allstates.dta*



Chapter 4
Bar graphs

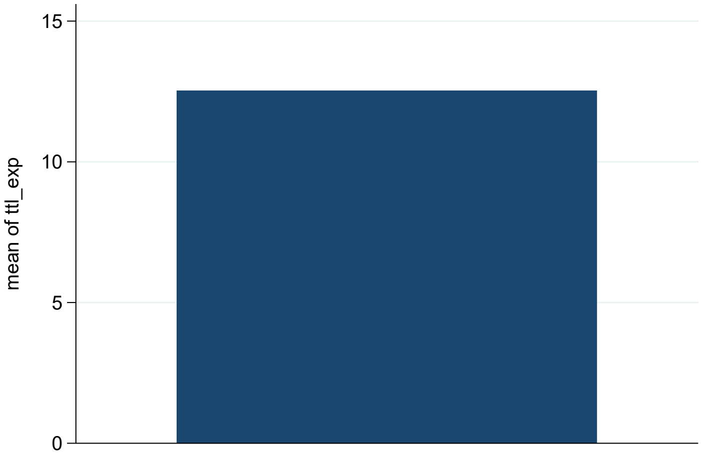
This chapter shows how to create bar charts using the `graph bar` command. It begins by showing how to use `graph bar` to create overall bar graphs of one or more continuous *y* variables. The next section illustrates how you can use the `over()` option to show the bars broken down by one, two, or three categorical variables. Next I illustrate the ways that you can control the gaps among bars, giving you fine control over the gaps between bars from each `over()` variable. Then I show how to control the sorting of bars, and then I illustrate how to customize the labels of the bars as well as the size and color of the labels. The next section illustrates how you can use the `legend()` option to customize the display of the bar graph legend, including how to modify the labels and position the legend. This section also illustrates how you can attach labels to bars that contain the bar label or the bar height (that is, the mean of the bar) and how you can customize those labels. Next I illustrate options allowing you to control the *y* axis, including options for customizing the title, customizing the labels, and even hiding the axis altogether. The next section illustrates how you can customize the colors of bars, and the last section illustrates how to use the `by()` option to create multiple graphs as a function of the levels of the *by-variable* specified in the `by()` option. For this entire chapter, we will use the `nlsw` data file.

4.1 Y variables

A bar chart graphs one or more continuous variables broken down by one or more categorical variables. The continuous variables are graphed on the y axis and are referred to as y variables. This section shows you how to specify the y variables using the `graph bar` command, how to include one or more y variables, and how to obtain different summary statistics for the y variables. For more information, see `help graph bar`.

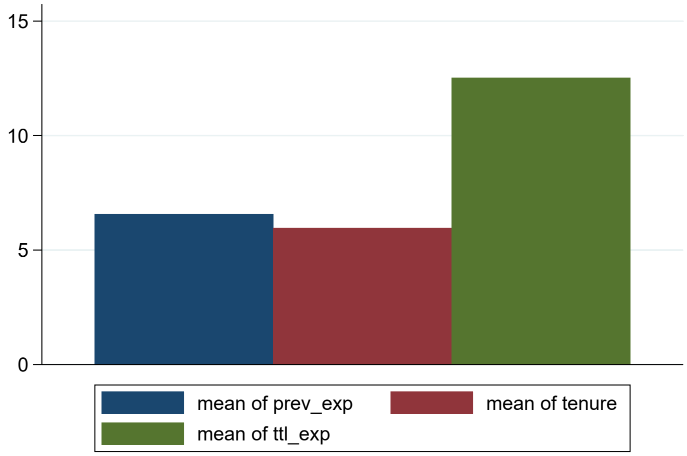
```
graph bar ttl_exp
```

This is probably the most basic bar chart that we can make (and perhaps the most boring, as well). It shows the average total work experience for all observations in the file. It graphs one y variable by using the default summary statistic, the mean. *Uses nlsw.dta*

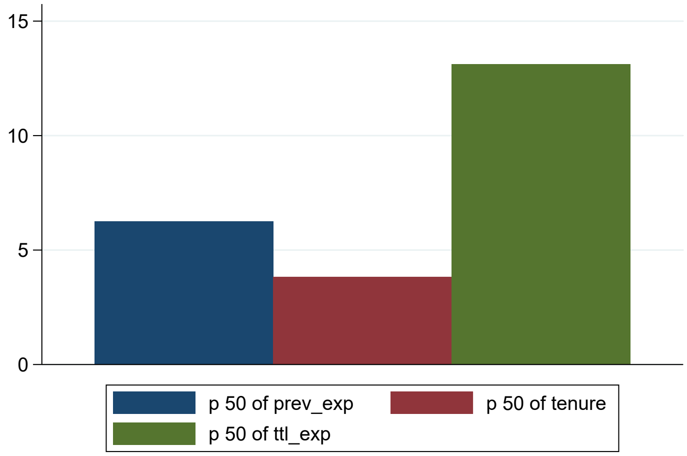


```
graph bar prev_exp tenure ttl_exp
```

We can specify multiple y variables to be plotted simultaneously. Here we graph the mean of previous, current, and total work experience in the same plot. The bars are plotted touching each other, and a legend indicates which bar corresponds to which variable. *Uses nlsw.dta*



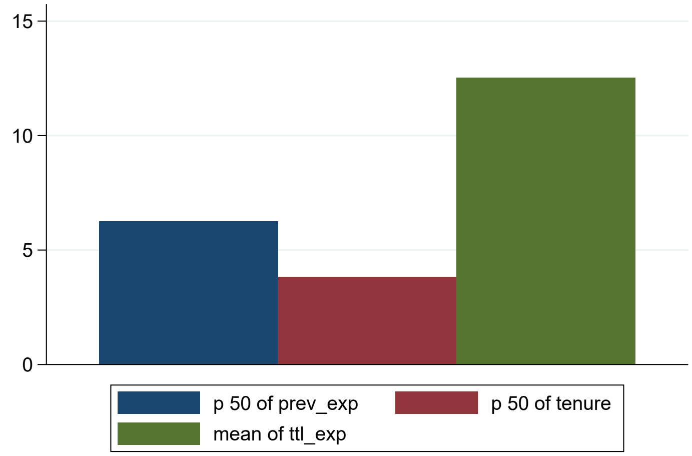
```
graph bar (median) prev_exp tenure ttl_exp
```



This graph is much like the last one, but it shows the median of these y variables. We only specified `(median)` before `prev_exp` but it applied to all the y variables that follow. You can summarize the y variables by using any of the summary statistics permitted by the `collapse` command (for example, `mean`, `sd`, `sum`, `median`, and `p10`); see `help collapse`. *Uses nlsw.dta*

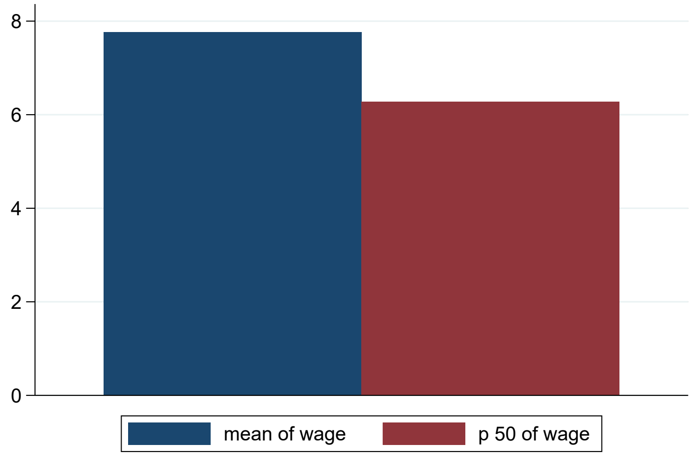
```
graph bar (median) prev_exp tenure (mean) ttl_exp
```

Here we get the median of the first two y variables and then the mean of the last y variable. *Uses nlsw.dta*



```
graph bar (mean) wage (median) wage
```

We can plot different summary statistics for the same y variable. In this example, we plot the mean and median wage. *Uses nlsw.dta*

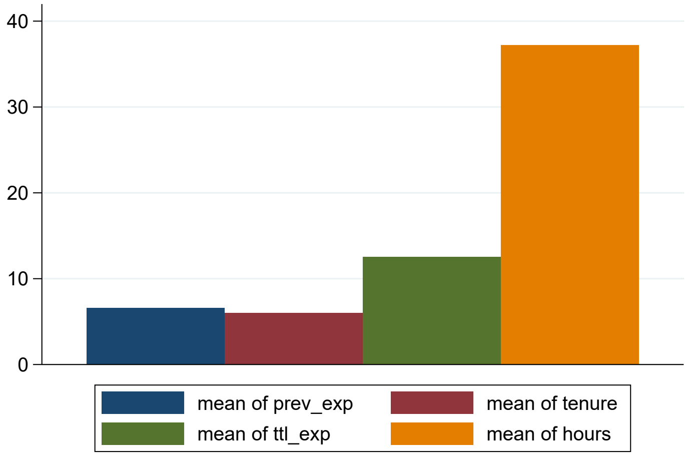


Let's now consider a handful of options that are useful when we have multiple y variables. These options allow us to display the y variables as though they were categories of the same variable, to create stacked bar charts, and to display the y variables as percentages of the total y variables.

```
graph bar prev_exp tenure ttl_exp hours
```

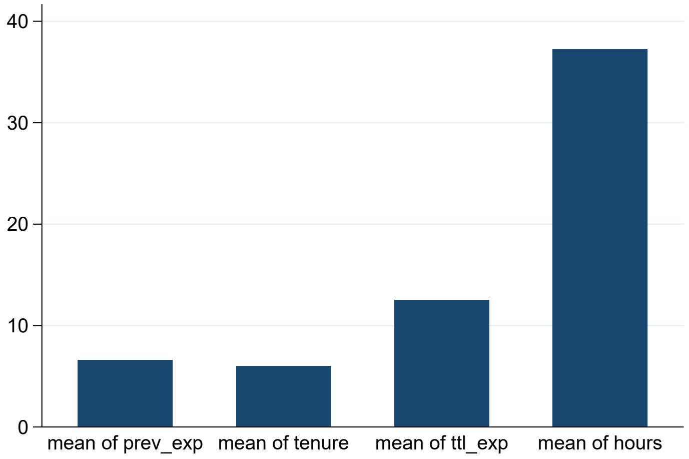
First, consider this bar chart showing four y variables. Each y variable is shown with a different colored bar and with a legend indicating which y

variable corresponds to which bar. See the next example for another way to differentiate these four bars. *Uses nlsw.dta*



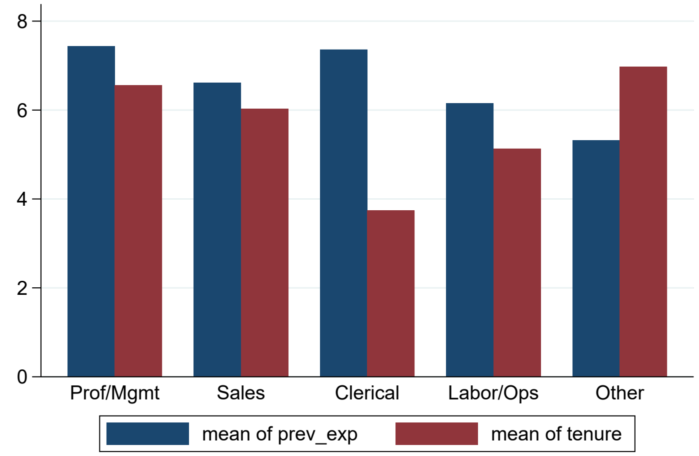
```
graph bar prev_exp tenure ttl_exp hours, ascategory
```

The `ascategory` option indicates that we want Stata to graph multiple y variables by using the style that would be used for the levels of an `over()` variable. Comparing this graph with the previous graph, note how the bars for the different variables are the same color and labeled on the x axis rather than using a legend. *Uses nlsw.dta*



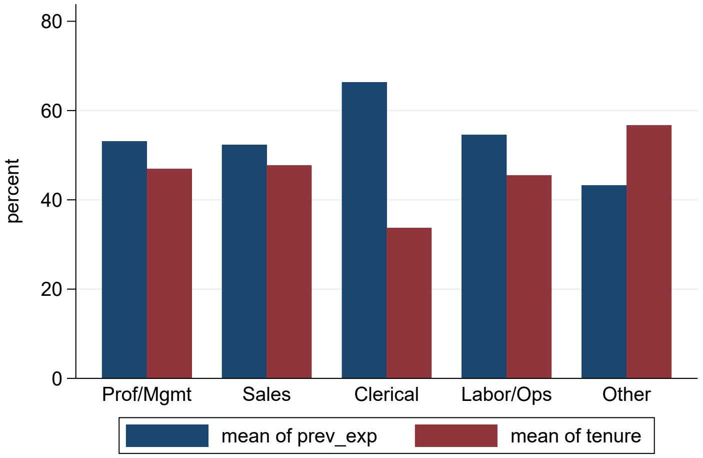
```
graph bar prev_exp tenure, over(occ5)
```

Consider this graph, where we show work experience prior to one's current job (`prev_exp`) and work experience at one's current job (`tenure`) broken down by `occ5`. The total of previous and current work experience represents total work experience, and we might want to show each bar as a percentage of total work experience. The next example shows how we can do that. *Uses nlsw.dta*



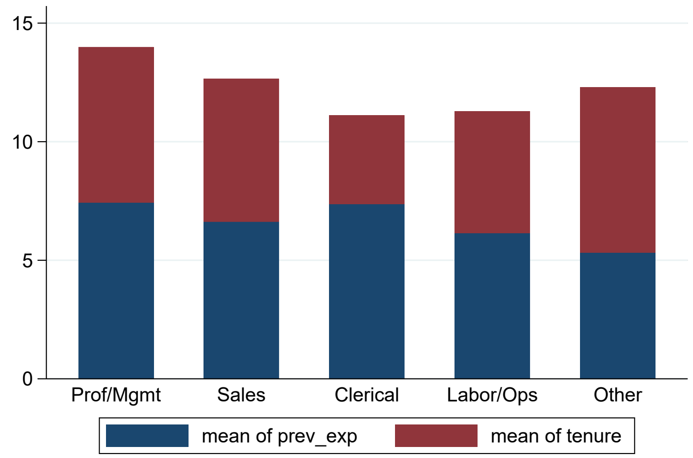
```
graph bar prev_exp tenure, over(occ5) percentages
```

Here we use the `percentages` option to show the time worked before one's current job, `prev_exp`, and the time at the current job, `tenure`, as a percentage of total work experience. *Uses nlsw.dta*



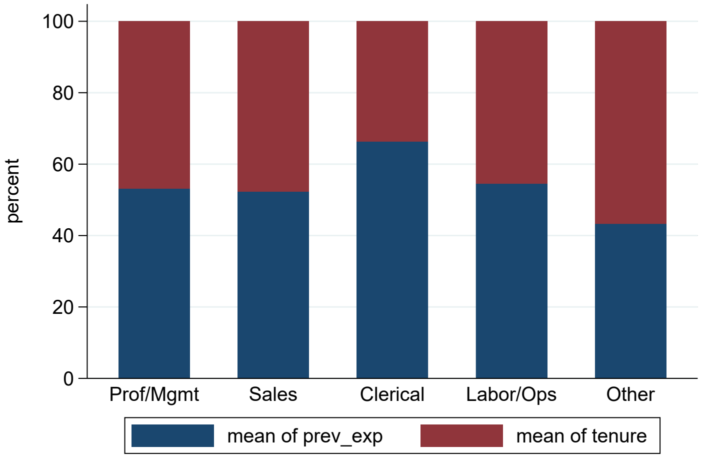
```
graph bar prev_exp tenure, over(occ5) stack
```

The `stack` option shows the y variables as a stacked bar chart. This option allows us to see the mean of each y variable, as well as the mean of the total y variables. *Uses nlsw.dta*



```
graph bar prev_exp tenure, over(occ5) percentages stack
```

We can also combine the `stack` and `percentages` options to create a stacked bar chart in terms of percentages. *Uses nlsw.dta*

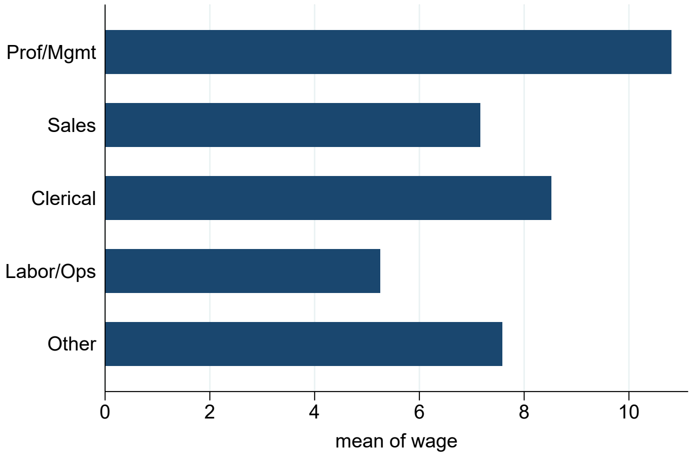


4.2 Graphing bars over groups

This section focuses on using the `over()` option for showing bar charts by one or more categorical variables. It illustrates using the `over()` option with one or multiple y variables. We will also look at some basic options, including those that do the following: display the `over()` variable as though its levels were multiple y variables, include missing values on the `over()` variable, and suppress empty combinations of multiple `over()` variables. See the *group_options* and *over_subopts* tables of `help graph bar` for more details.

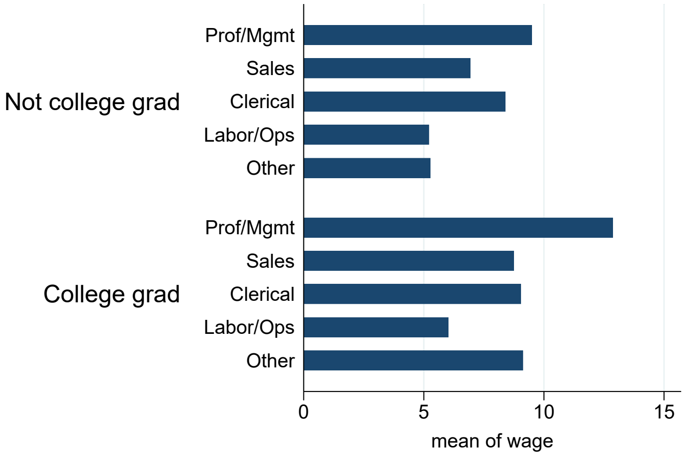
```
graph hbar wage, over(occ5)
```

In this example, we use the `over()` option to show the average wages broken down by occupation. We are using `graph hbar` to produce horizontal, rather than vertical, bar charts. *Uses nlsw.dta*

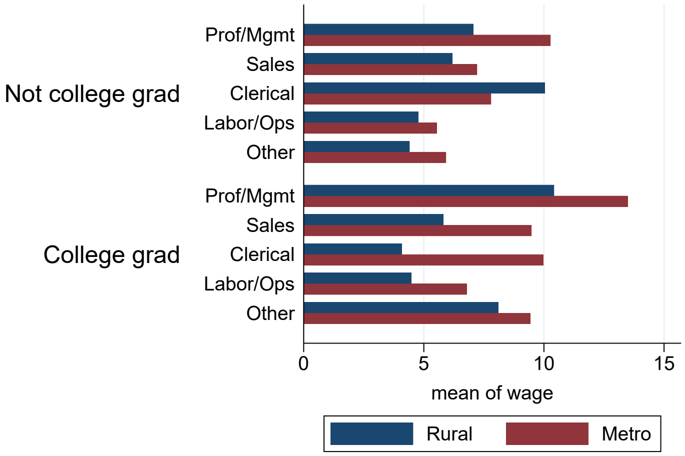


```
graph hbar wage, over(occ5) over(collgrad)
```

Here we use the `over()` option twice to show the wages broken down by occupation and whether one graduated college. Note the appropriate way to produce this graph is to use two `over()` options, rather than using one `over()` option with two variables. As we will see later, each `over()` can have its own options, allowing us to customize the display of each `over()` variable. *Uses nlsw.dta*



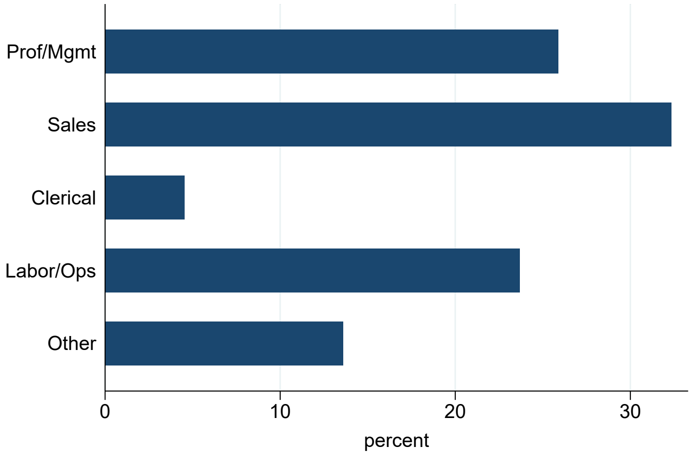
```
graph hbar wage, over(urban2) over(occ5) over(collgrad)
```



We can even add a third `over()` option, here, using `over(urban2)` to compare those living in rural versus urban areas. The look of the graph changes when we add the third `over()` variable. This is because Stata is now treating the first `over()` variable as though it were multiple y variables. Because of this, we can only specify one y variable when we have three `over()` options. *Uses nlsw.dta*

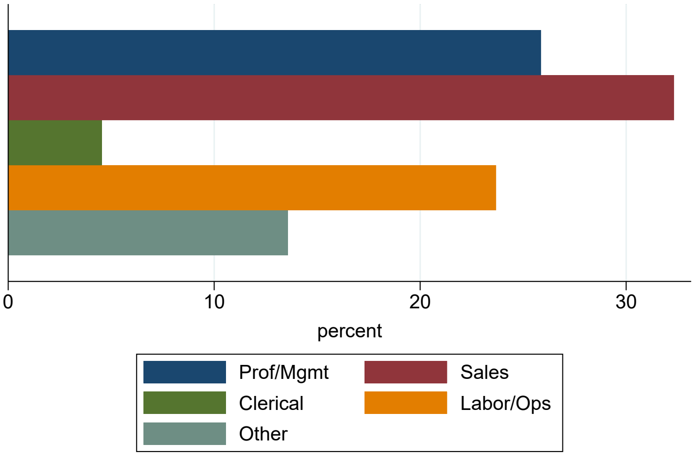
The prior examples illustrated how to create bar graphs of the average wages broken down by grouping variables (such as occupation or whether one graduated from college). The following examples illustrate how you can construct bar charts where the height of the bars represent the percentage of observations that fall within each category (such as occupation or whether one graduated from college).

```
graph hbar (percent), over(occ5)
```



This bar graph shows the percentage of observations that belong to each of the five occupations. For example, over 30% of workers in this sample worked in Sales. By specifying the `over(occ5)` option, we break down the results by `occ5`, and by specifying `(percent)` after `graph hbar`, we produce the percentage of observations within each group. *Uses nlsw.dta*

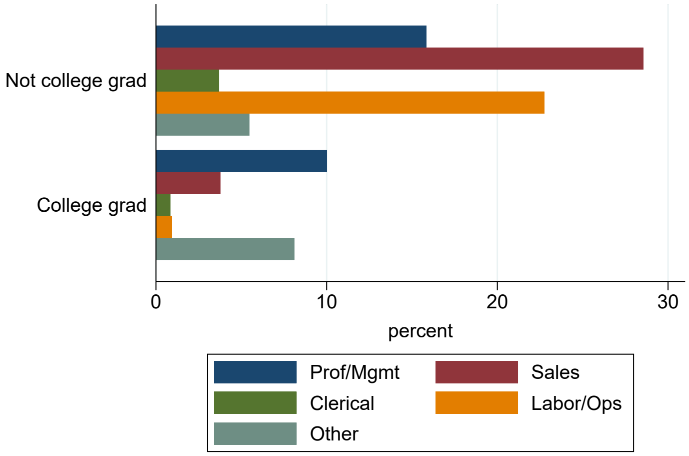
```
graph hbar (percent), over(occ5) asyvars
```



By adding the `asyvars` option, we display each level of `occ5` using different colors, and a legend is used to label each group. This is the style of output used when multiple variables are graphed at once, explaining the behavior of the option `asyvars`. *Uses nlsw.dta*

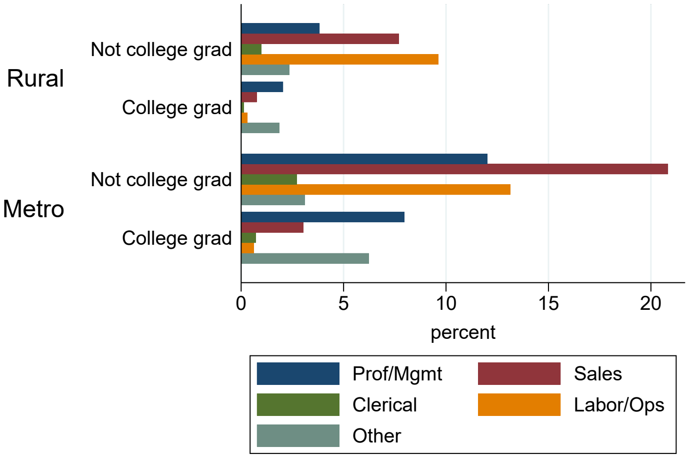
```
graph hbar (percent), over(occ5) over(collgrad) asyvars
```

This example includes two `over()` variables. The percentages graphed mirror the percentages we would obtain if we performed a cross-tabulation of `occ5` by `collgrad` and computed the cell percentages. For example, 10% of all workers were college graduates working in the category *Prof/Mgmt*. *Uses nlsw.dta*



```
graph hbar (percent), over(occ5) over(collgrad) over(urban2)
```

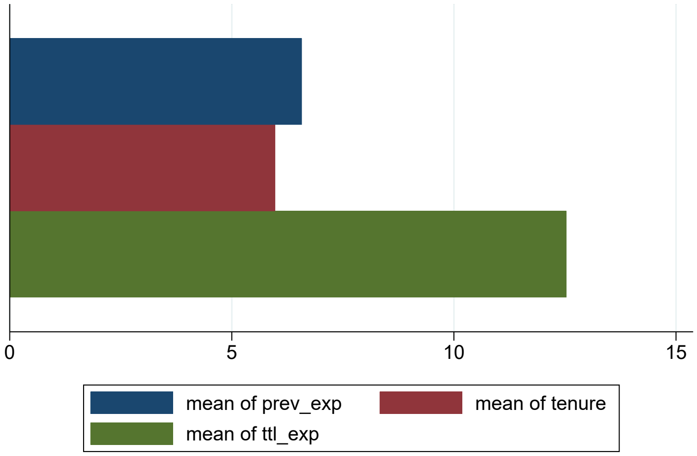
This example includes three `over()` variables. The bars represent the cell percentages from a three-way cross-tabulation of `occ5` by `collgrad` by `urban2`. For example, about 21% of all workers are noncollege graduates working in *Sales* and living in metropolitan areas. Because of the third `over()` option, the levels of the first `over()` variable are treated as multiple y variables. *Uses nlsw.dta*



Now let's look at examples of using multiple y variables with the `over()` option. Let's start with a simple bar graph with multiple y variables. These examples will use the `scheme(vg_lgndc)` option, which places the legend to the left of the graph and displays it in one, stacked column. See section 1.4.3 for more details about this scheme.

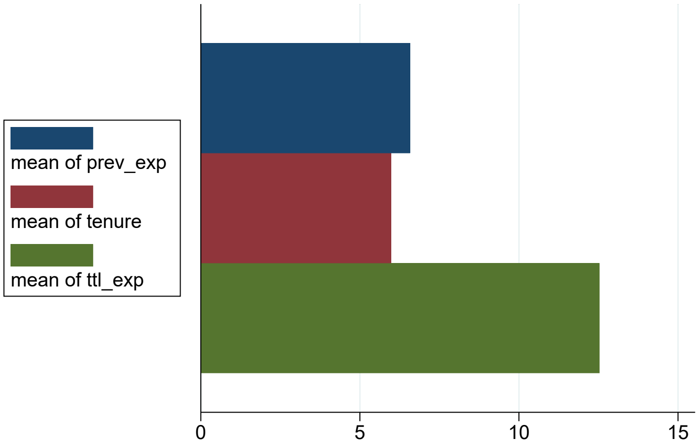
```
graph hbar prev_exp tenure ttl_exp
```

This graph shows the overall mean of previous, current, and total work experience. *Uses nlsw.dta*



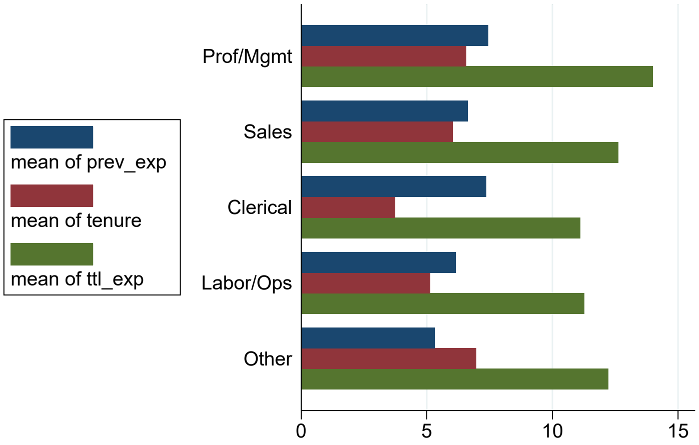
```
graph hbar prev_exp tenure ttl_exp, scheme(vg_lgndc)
```

This graph is the same as the prior graph but adds the `scheme(vg_lgndc)` option. Note how the legend is repositioned and reformatted by using this option. *Uses nlsw.dta*



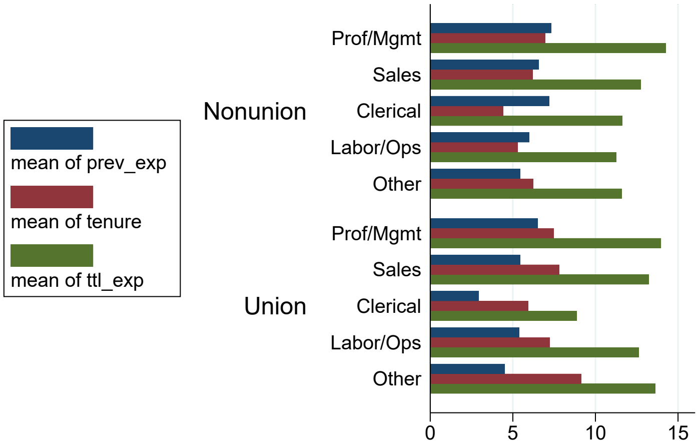
```
graph hbar prev_exp tenure ttl_exp, over(occ5)
scheme(vg_lgndc)
```

We can take the graph from above and break down the means by occupation by adding the `over(occ5)` option. *Uses nlsw.dta*



```
graph hbar prev_exp tenure ttl_exp, over(occ5) over(union)
scheme(vg_lgndc)
```

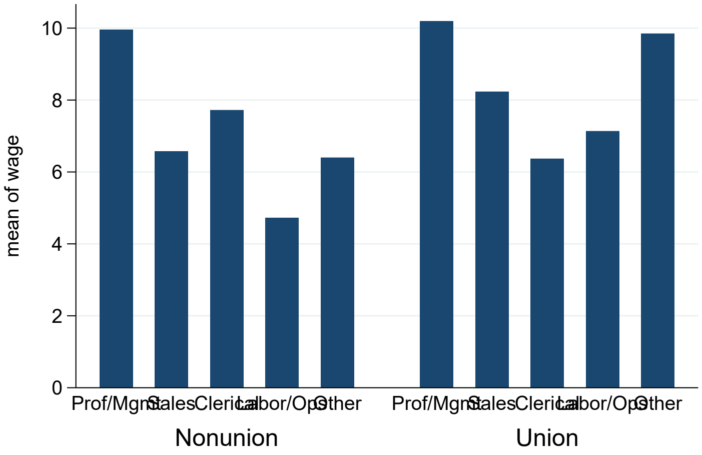
We can take the previous graph and further break down the results by whether one belongs to a union. However, we cannot add a third `over()` option when we have multiple y variables. *Uses nlsw.dta*



Now let’s consider options we can use in combination with the `over()` option to customize the graphs. These examples will illustrate how we can treat the levels of the variable in the first `over()` option as though they were multiple y variables and can even graph those levels as percentages or stacked bar charts. We can also request that missing values for the levels of the `over()` variables be displayed, and we can suppress empty levels when multiple `over()` options are used.

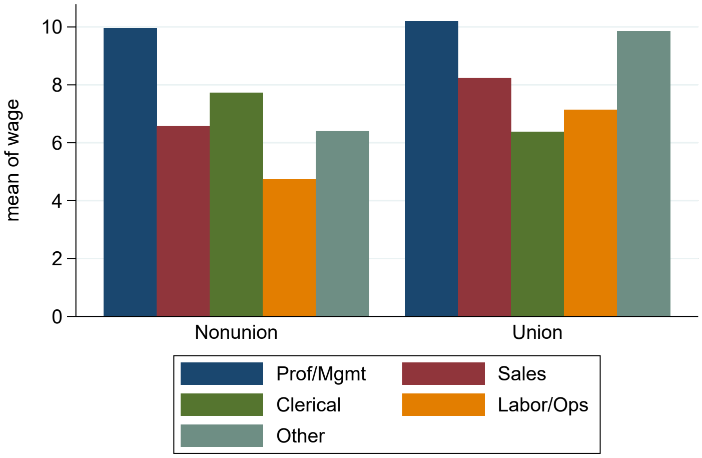
```
graph bar wage, over(occ5) over(union)
```

Consider this graph, where we show wages broken down by occupation and whether one belongs to a union. The labels for the levels of `occ5` overlap, but this is corrected in the next example. *Uses nlsw.dta*



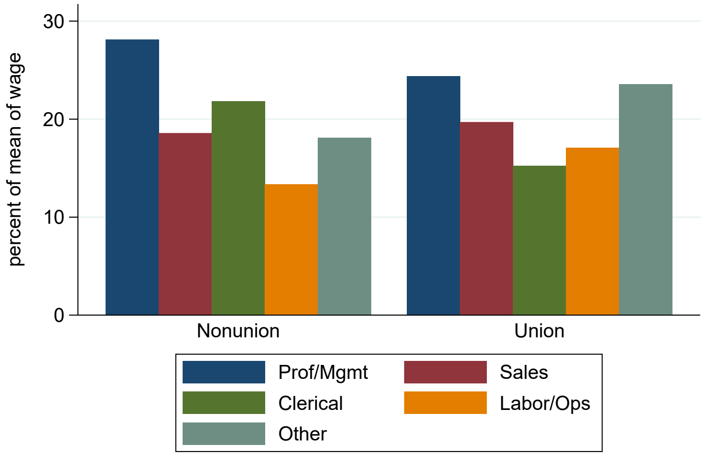
```
graph bar wage, over(occ5) over(union) asyvars
```

If we add the `asyvars` option, the first `over()` variable (`occ5`) is graphed as if there were five y variables corresponding to the five levels of `occ5`. The levels of `occ5` are shown as differently colored bars pushed next to each other and labeled using the legend. *Uses nlsw.dta*



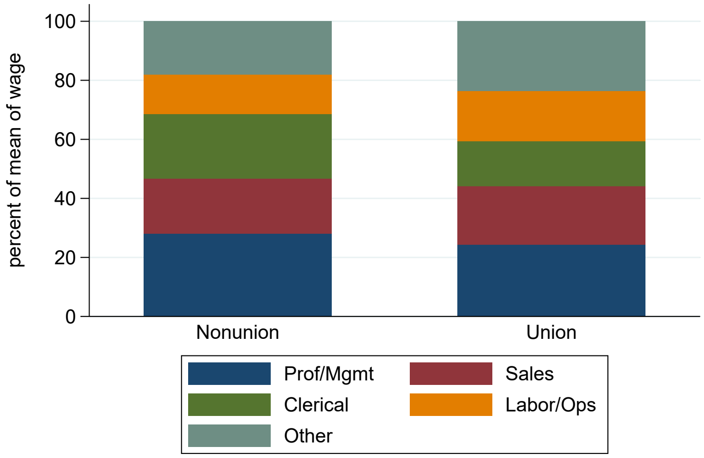
```
graph bar wage, over(occ5) over(union) asyvars percentages
```

With the levels of `occ5` considered as y variables, we can use some of the options that apply when we have multiple y variables. Here we request that the values be plotted as `percentages`. *Uses nlsw.dta*



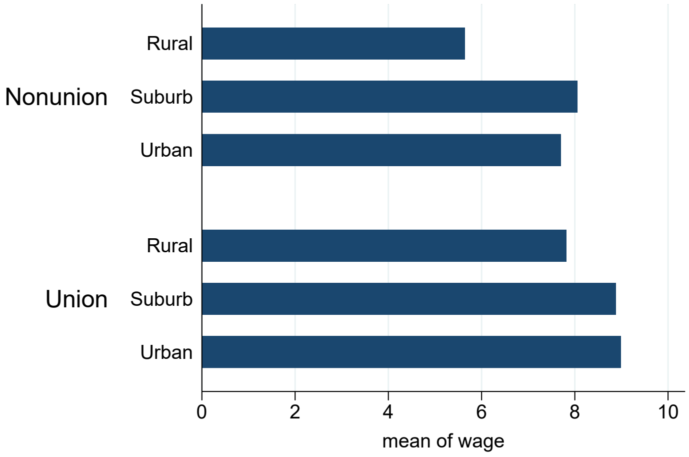
```
graph bar wage, over(occ5) over(union) asyvars percentages
stack
```

Again, because we are treating the levels of `occ5` as though they were multiple y variables, we can add the `stack` option to view the graph as a stacked bar chart. *Uses nlsw.dta*

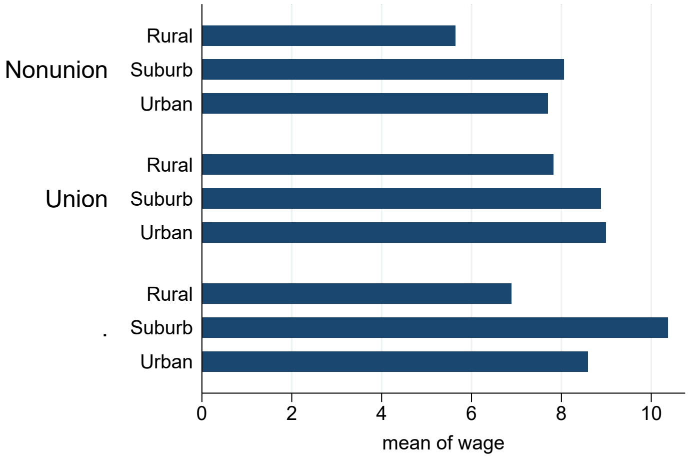


```
graph hbar wage, over(urban3) over(union)
```

Consider this graph, where we use the `over(union)` option to compare the mean wages of union workers with nonunion workers. One aspect this graph hides is that there are several missing values on the variable `union`. *Uses nlsw.dta*



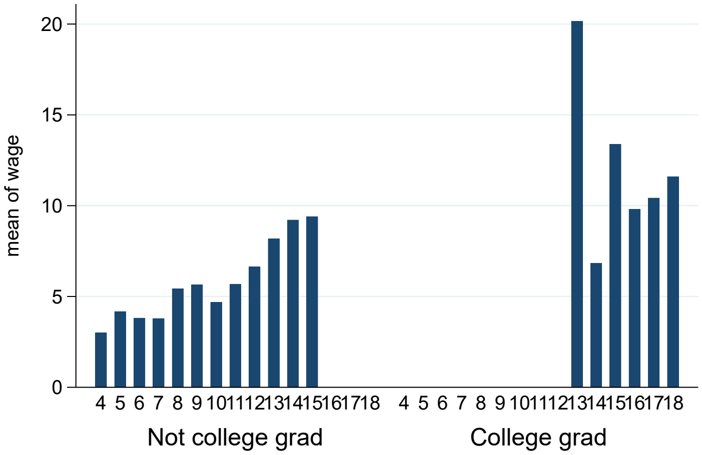
```
graph hbar wage, over(urban3) over(union) missing
```



By adding the `missing` option, we then see a category for those who are missing on the `union` variable, shown as the third set of bars. The label for this bar is one dot, which is the Stata indicator of missing values. The section Bar : Cat axis (section 4.5) shows how we can give this bar a more meaningful label. *Uses nlsw.dta*

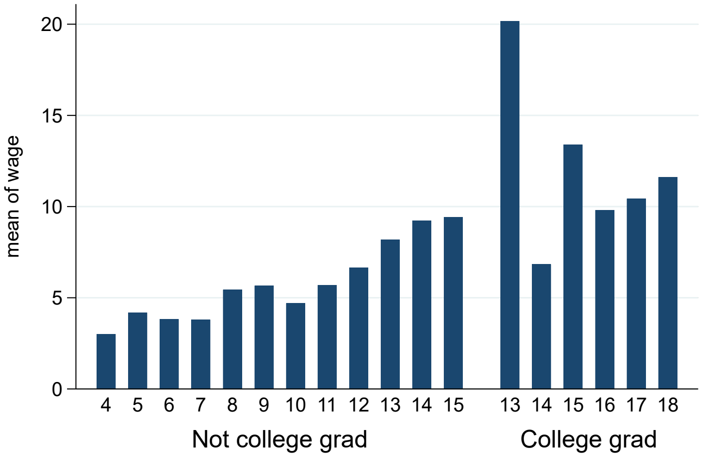
```
graph bar wage, over(grade) over(collgrad)
```

Consider this bar chart, which breaks down wages by two variables: the last grade that one completed and whether one is a college graduate. By default, Stata shows all possible combinations for these two variables. Usually, all combinations are possible, but not here. *Uses nlsw.dta*



```
graph bar wage, over(grade) over(collgrad) nofill
```

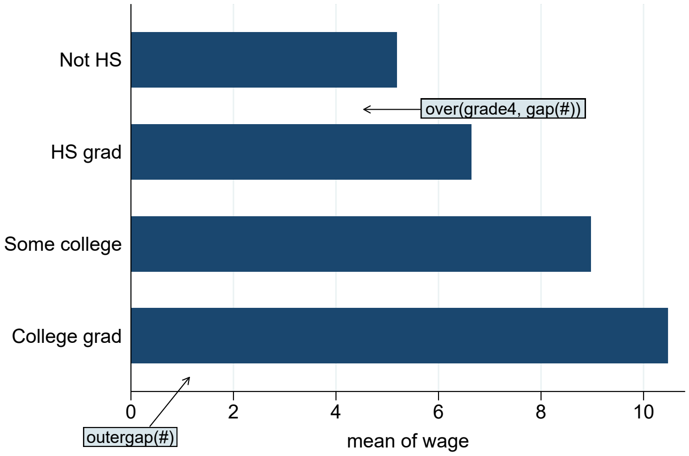
If we want to display only the combinations of the `over()` variables that exist in the data, we use the `nofill` option. *Uses nlsw.dta*



4.3 Options for controlling gaps between bars

This section illustrates how to control gaps between bars and the gaps outside the bars. There are no options to control the actual width of the bars. These are controlled by specifying the size of the gaps, and whatever is leftover determines the width of the bars (that is, specifying big gaps yields skinny bars, and specifying small gaps yields wide bars).

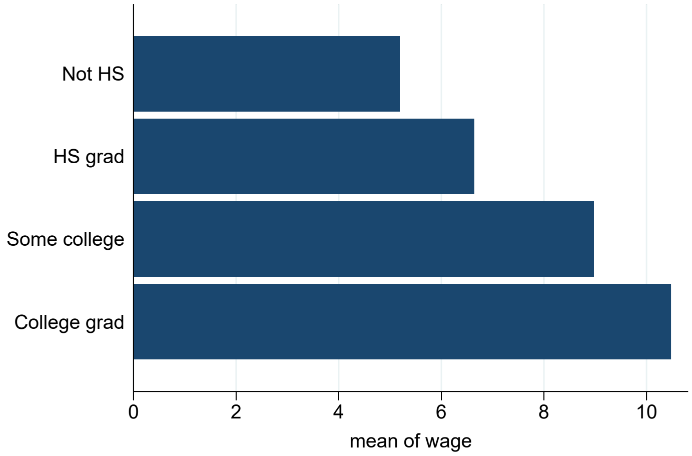
```
graph hbar wage, over(grade4)
```



We start with a simple graph showing the average wages broken down by `grade4`. This graph is annotated using the Graph Editor to show the gaps that we can control. The default gap between levels of `grade4` bars (controlled by `over(grade4, gap(#))` is 67% of the width of a bar. The default value for the `outergap(#)` is 20% the width of a bar. The following examples will illustrate altering these defaults. *Uses nlsw.dta*

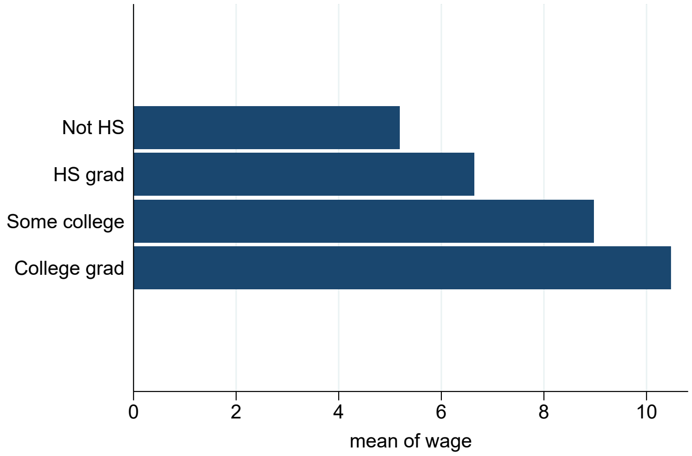
```
graph hbar wage, over(grade4, gap(10))
```

Here we add the `gap(10)` option to make the gap between the bars be 10% of the width of a bar. *Uses nlsw.dta*



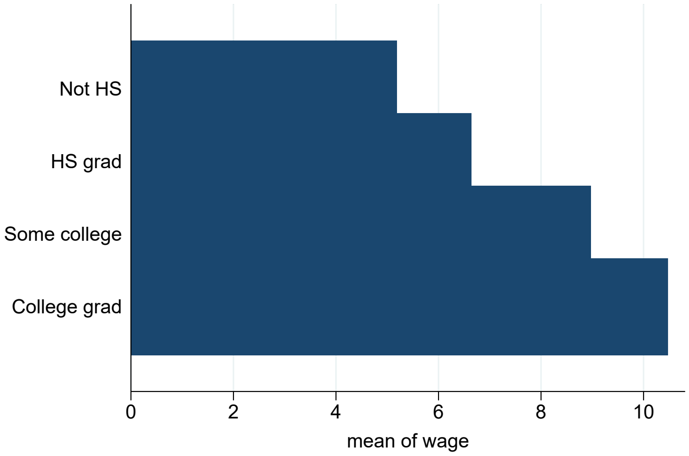
```
graph hbar wage, over(grade4, gap(10)) outergap(200)
```

We now add the `outergap(200)` option to increase the outer gap between the peripheral bar and the edge of the graph from 20% of the width of a bar to 200% times the width of a bar. *Uses nlsw.dta*



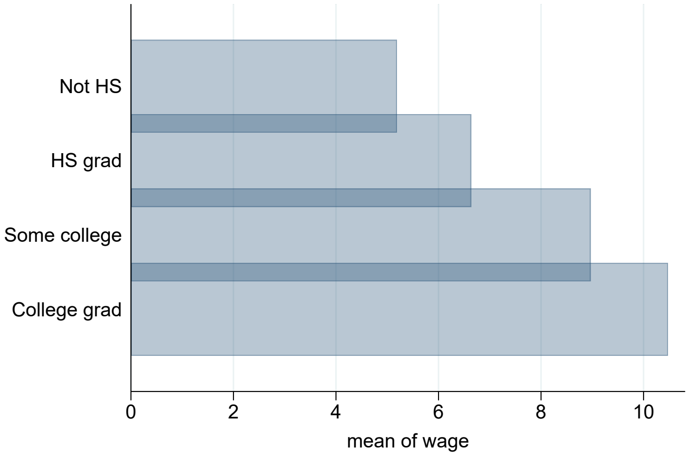
```
graph hbar wage, over(grade4, gap(-25))
```

Specifying a negative gap might seem illogical, but this is how to indicate that you want the bars to overlap. Specifying −25 indicates that the bars should overlap by 25% of the width of a bar. *Uses nlsw.dta*



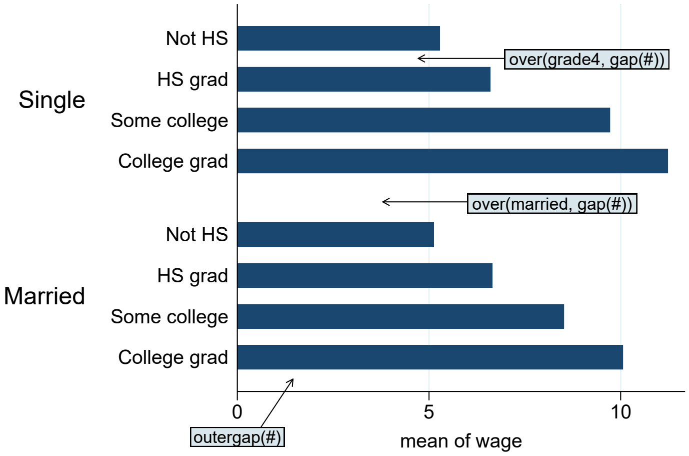
```
graph hbar wage, over(grade4, gap(-20)) bar(1, bcolor(%30))
```

I think the previous graph looks better by reducing the opacity of the bars. This example specifies `bar(1, bcolor(%30))`, making the opacity of the bars 30%. *Uses nlsw.dta*



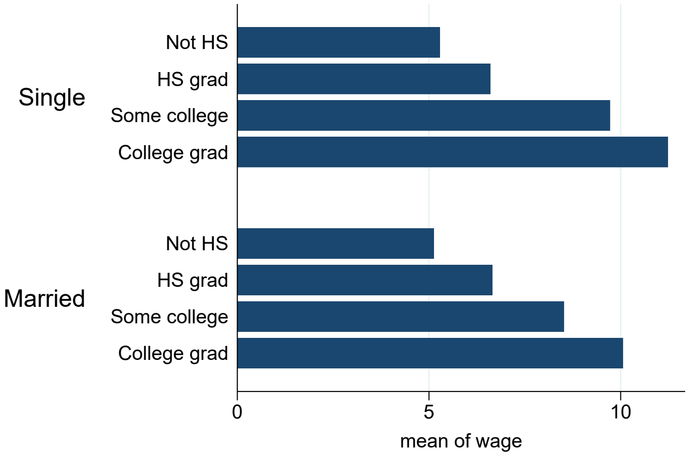
```
graph hbar wage, over(grade4) over(married)
```

We now look at a graph with two `over()` options. This graph is annotated using the Graph Editor to show the gaps that we can control. The default gap between levels of `grade4` bars (controlled by `over(grade4, gap(#))` is 67% of the width of a bar, whereas the default gap between the levels of `married` is 200% times the width of a bar. The default value for the `outergap(#)` is 20% the width of a bar. The following examples will illustrate altering these defaults. *Uses nlsw.dta*



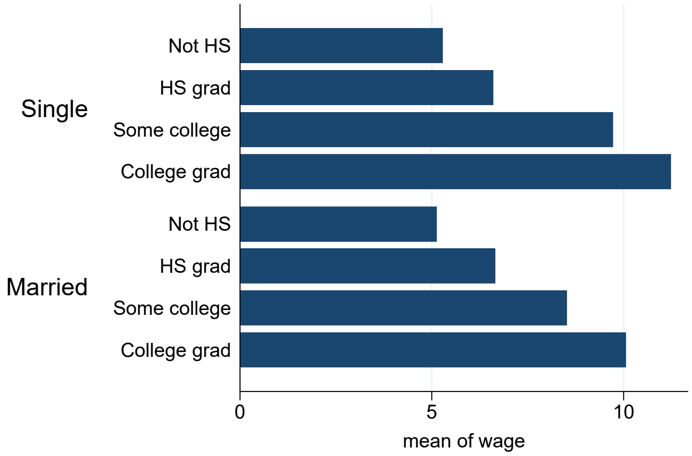
```
graph hbar wage, over(grade4, gap(20)) over(married)
```

By adding the `gap(20)` option, we shrink the gap among the education categories from 67% to 20%. *Uses nlsw.dta*



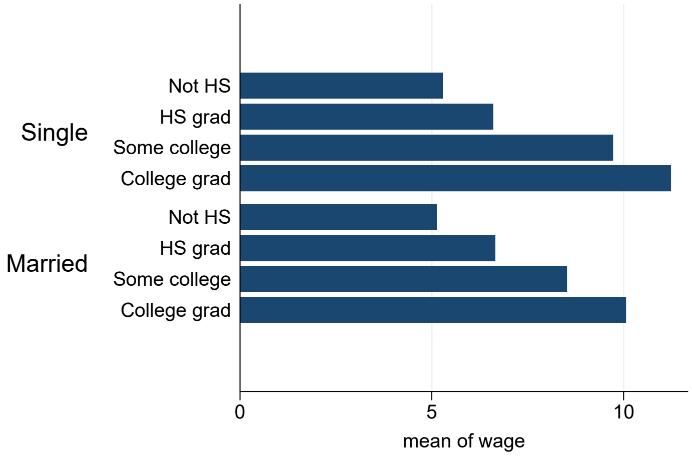
```
graph hbar wage, over(grade4, gap(20)) over(married, gap(50))
```

The gap between single and married would normally be 200% times the width of a bar, but here we reduce it to 50%. *Uses nlsw.dta*



```
graph hbar wage, over(grade4, gap(20)) over(married, gap(50))
outergap(200)
```

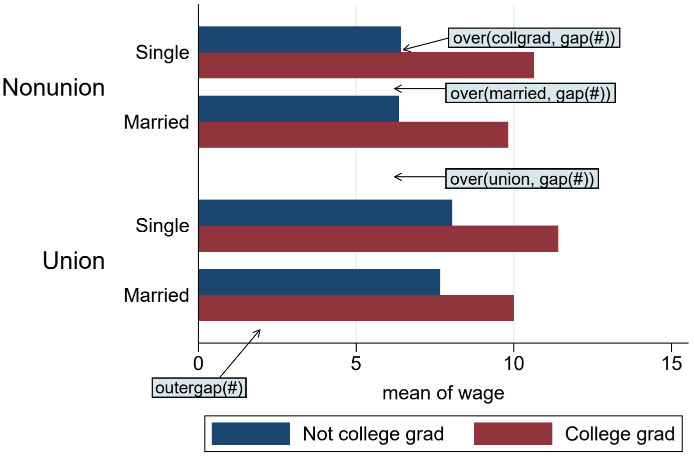
The outer gap is normally 20% of the width of a bar, but here we expand it to 200% times the width of a bar by specifying the `outergap(200)` option. *Uses nlsw.dta*



```
graph hbar wage, over(collgrad) over(married) over(union)
```

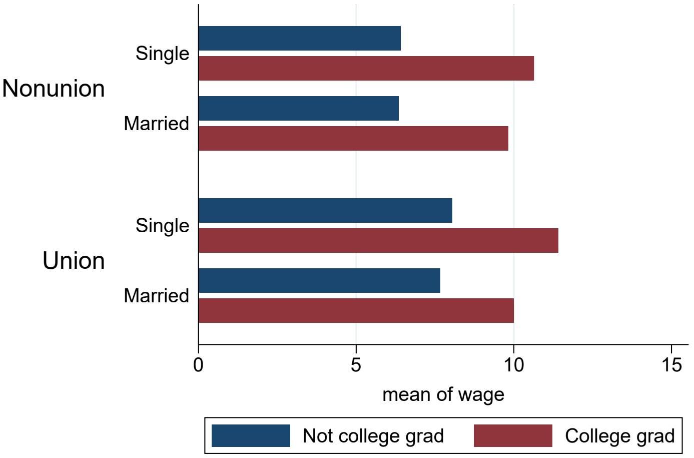
We now look at a graph with three `over()` options. This graph is annotated using the Graph Editor to show the gaps that we can control. The default gap between levels of `collgrad` bars (controlled by `over(collgrad, gap(#))` is zero. The default gap between levels of married is 67% of the width of a bar, whereas the default gap between the levels of `union` is 200% times the width of a bar. The default value for the `outergap(#)` is

20% the width of a bar. The following examples will illustrate altering these defaults. *Uses nlsw.dta*



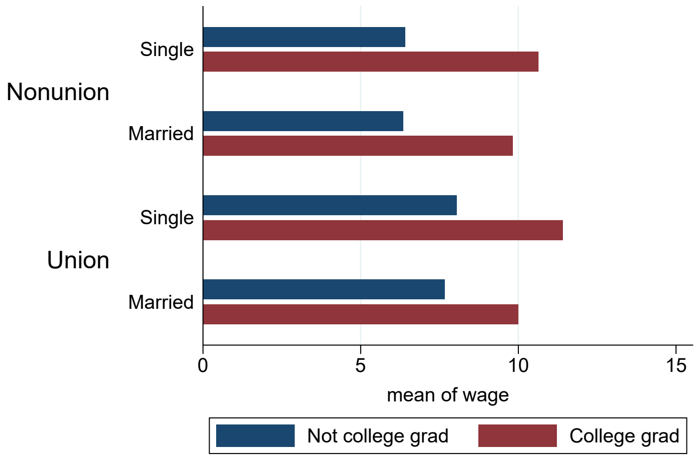
```
graph hbar wage, over(collgrad, gap(25)) over(married)
over(union)
```

Normally, the gap among the levels of `collgrad` would be 0%, but here we make the gap 25% of the width of a bar. *Uses nlsw.dta*



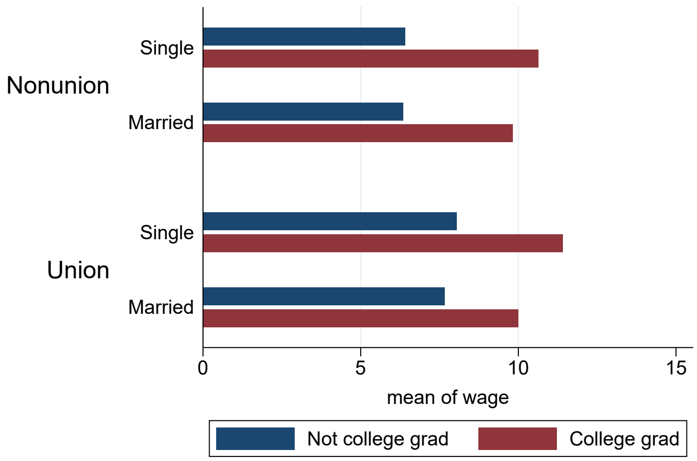
```
graph hbar wage, over(collgrad, gap(25)) over(married,
gap(200)) over(union)
```

Normally, the gap between single and married would be two-thirds the width of a bar (67%), but here we increase that gap to 200%. *Uses nlsw.dta*



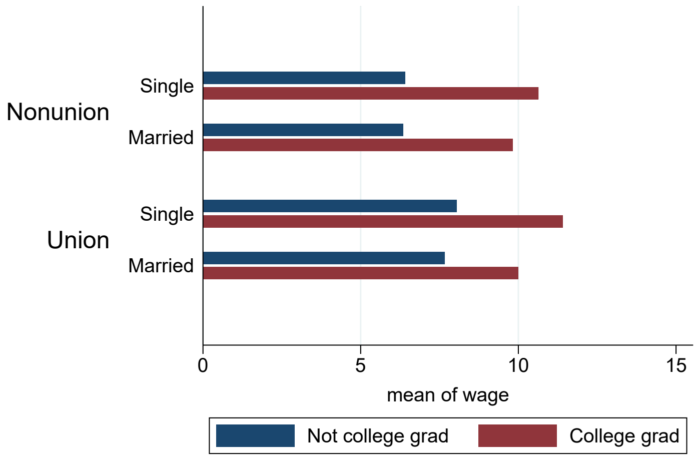
```
graph hbar wage, over(collgrad, gap(25)) over(married,
gap(200)) over(union, gap(400))
```

The gap between union and nonunion would normally be twice the width of a bar (200%), but here we increase it to 400%. *Uses nlsw.dta*

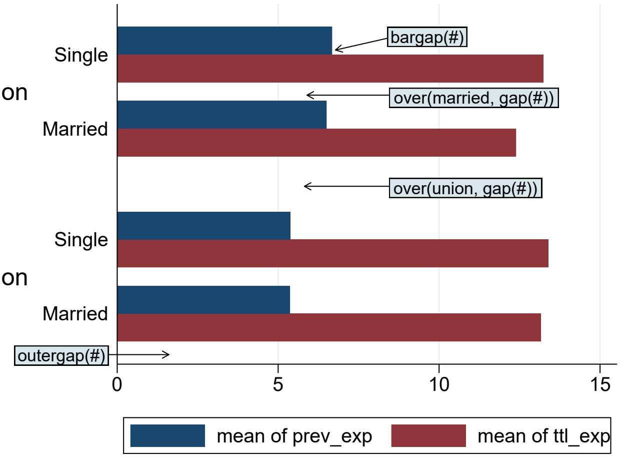


```
graph hbar wage, over(collgrad, gap(25)) over(married,
gap(200)) over(union, gap(400)) outergap(400)
```

The outer gap is normally one-fifth (20%) of the width of a bar, but here we increase it to four times the width of a bar (400%). *Uses nlsw.dta*



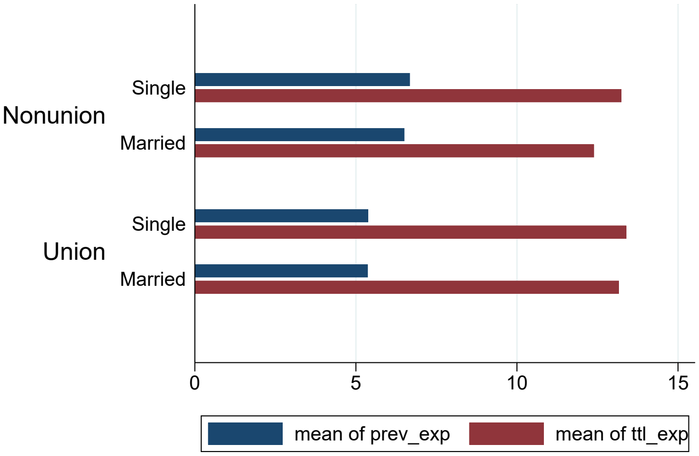
```
graph hbar prev_exp ttl_exp, over(married) over(union)
```



The final examples of this section look at a graph that has multiple y variables (here, two) along with two `over()` options. This graph is annotated using the Graph Editor to show the gaps that we can control. The default gaps are all the same as the previous example, but because this graph involves multiple y variables, the gap between `prev_exp` and `ttl_exp` is controlled by the `bargap()` option, as illustrated in the next example. *Uses nlsw.dta*

```
graph hbar prev_exp ttl_exp, bargap(25) over(married,
gap(200)) over(union, gap(400)) outergap(400)
```

Here we use the `bargap()` option to make the gap between `prev_exp` and `ttl_exp` 25% of the width of a bar. We also see that the `gap()` and `outergap()` options work as they did in the previous examples. *Uses nlsw.dta*

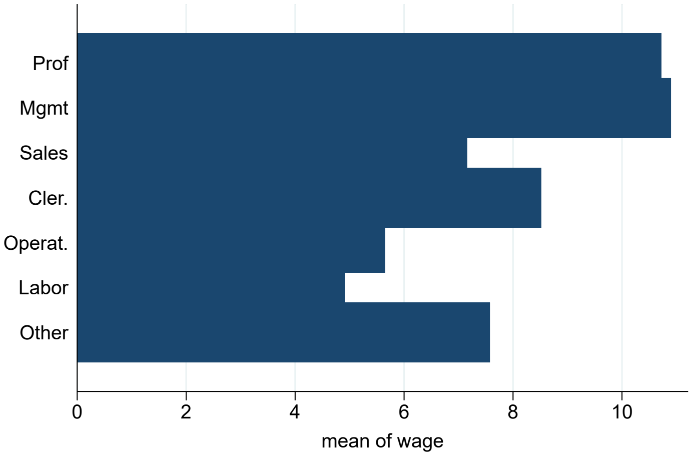

Nonunion
Single
Married
Union
Single
Married
0
5
10
15
mean of prev_exp
mean of ttl_exp

4.4 Options for sorting bars

This section considers some of the options we can use to control the order in which the bars are displayed. By default, the bars formed by `over()` variables are ordered in ascending sequence according to the values of the `over()` variable. However, Stata gives us considerable flexibility in the ordering of the bars.

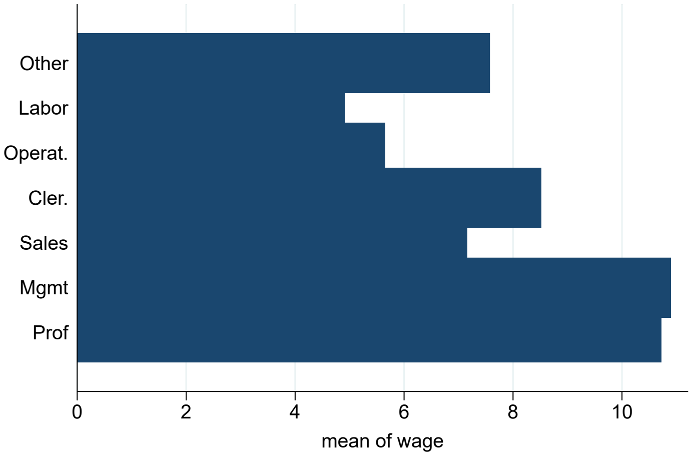
```
graph hbar wage, over(occ7, gap(-25))
```

Let's start with this graph showing the average wages broken down by occupation. The bars are ordered by the levels of `occ7`, going from 1 to 7, where 1 is *Prof* and 7 is *Other*. The first bar (*Prof*) overlaps the second bar (*Mgmt*), and the second bar overlaps the third (*Sales*), and so forth. *Uses nlsw.dta*



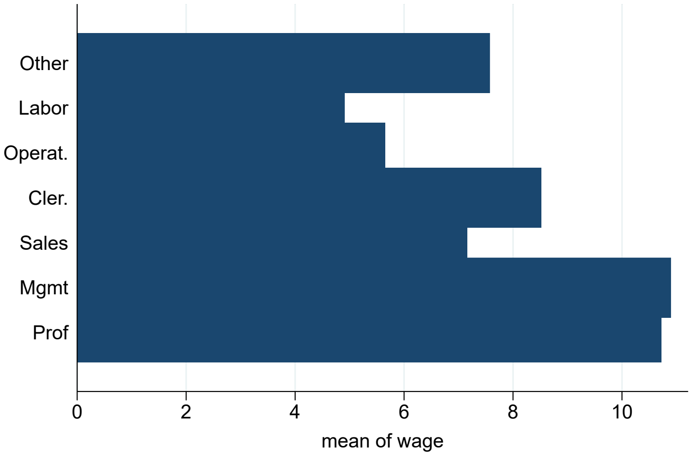
```
graph hbar wage, over(occ7, gap(-25) descending)
```

By using the `descending` option, the order of the bars is switched. The bars are still ordered according to the seven levels of occupation, but they are ordered going from 7 to 1. *Uses nlsw.dta*



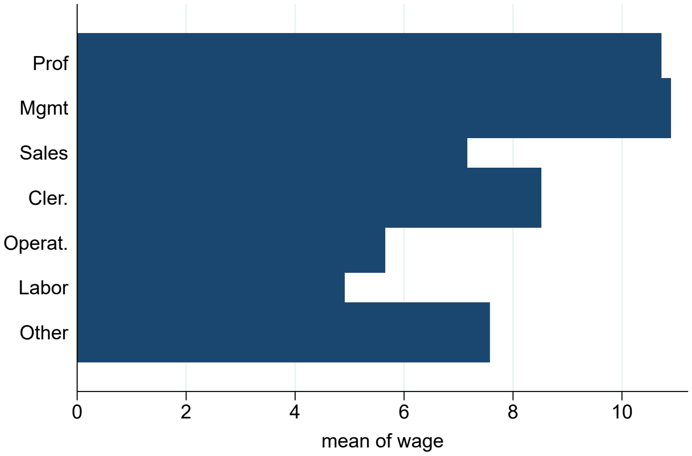
```
graph hbar wage, over(occ7, gap(-25) reverse)
```

Using the `reverse` option not only switches the order of the bars, but it also changes the order of overlapping. Here *Prof* overlaps *Mgmt*, whereas the overlapping was the opposite in the previous graph. *Uses nlsw.dta*

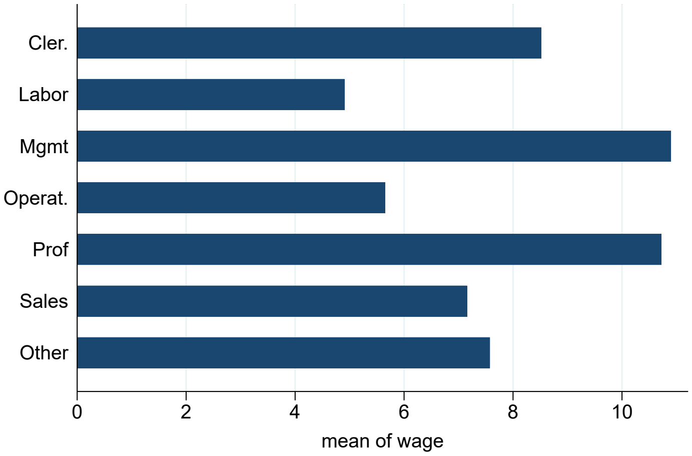


```
graph hbar wage, over(occ7, gap(-25) reverse descending)
```

Although it might seem counterintuitive, we can combine the `reverse` and `descending` options. They counteract each other with respect to the ordering of the categories (the categories are ordered from 1 to 7), but the net effect is that the pattern of overlapping is reversed. In this graph *Mgmt* overlaps *Prof*, compared with the first graph in this section, where *Prof* overlaps *Mgmt*. *Uses nlsw.dta*



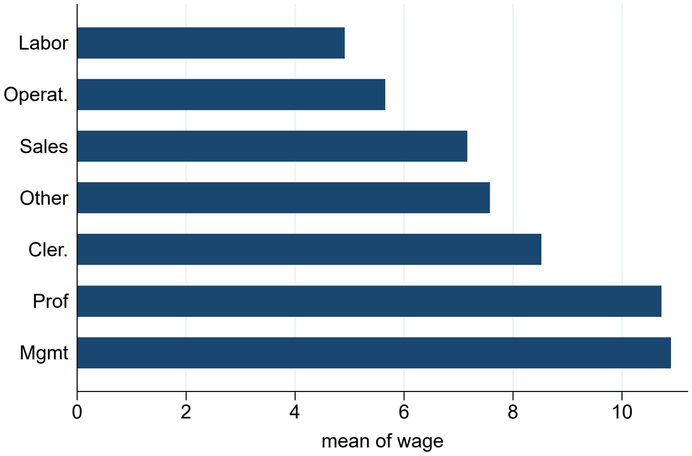
```
graph hbar wage, over(occ7, sort(occ7alpha))
```



We might want to put these bars in alphabetical order (but with *Other* still appearing last). We can do this by recoding `occ7` into a new variable (say, `occ7alpha`) such that as `occ7alpha` goes from 1 to 7, the occupations are alphabetical. We recoded `occ7` with these assignments: $4 = 1$, $6 = 2$, $2 = 3$, $5 = 4$, $1 = 5$, $3 = 6$, and $7 = 7$; see `help recode`. Then the `sort(occ7alpha)` option has the effect of alphabetizing the bars. *Uses nlsw.dta*

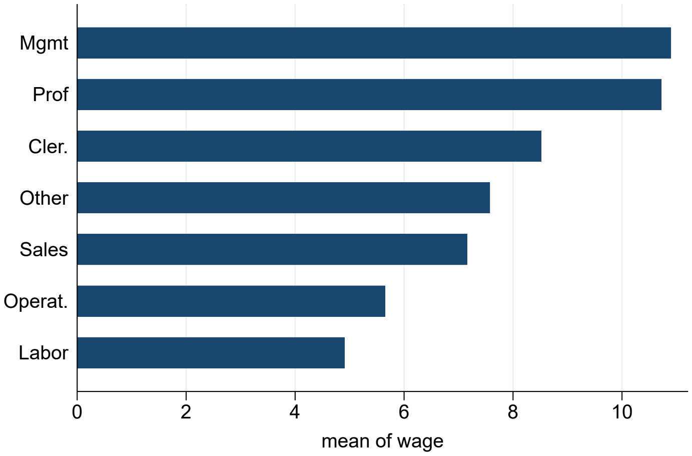
```
graph hbar wage, over(occ7, sort(1))
```

Here we sort the variables on the height of the bars (in ascending order). The `sort(1)` option sorts the bars according to the height of the first y variable, the mean of `wage`. *Uses nlsw.dta*



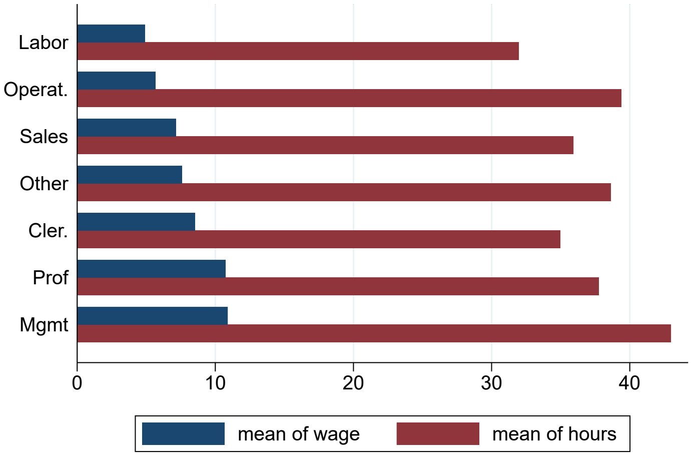
```
graph hbar wage, over(occ7, sort(1) descending)
```

Adding the `descending` option yields bars in descending order. *Uses nlsw.dta*



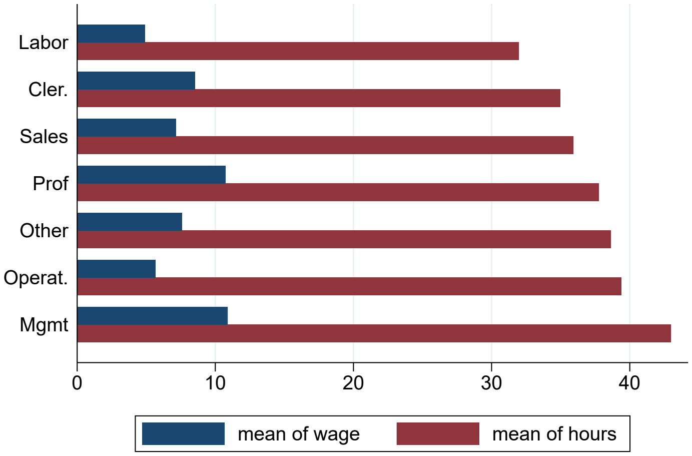
```
graph hbar wage hours, over(occ7, sort(1))
```

Here we plot two y variables. In addition to wages, we show the average hours worked per week. The `sort(1)` option sorts the bars according to the mean of `wage` because that is the first y variable. *Uses nlsw.dta*



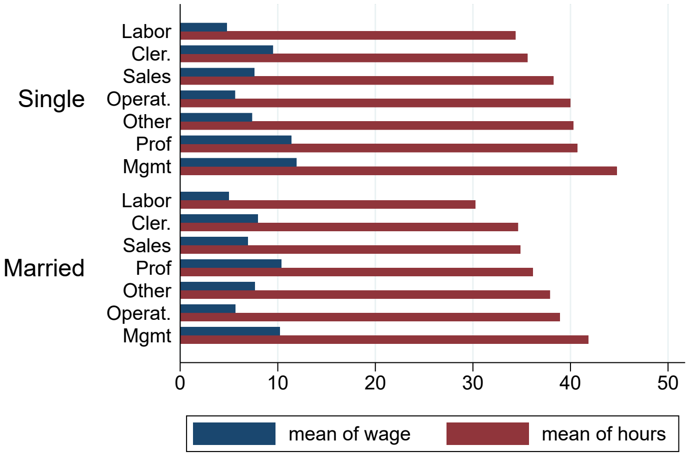
```
graph hbar wage hours, over(occ7, sort(2))
```

Changing `sort(1)` to `sort(2)` sorts the bars according to the second y variable, the mean of `hours`. *Uses nlsw.dta*



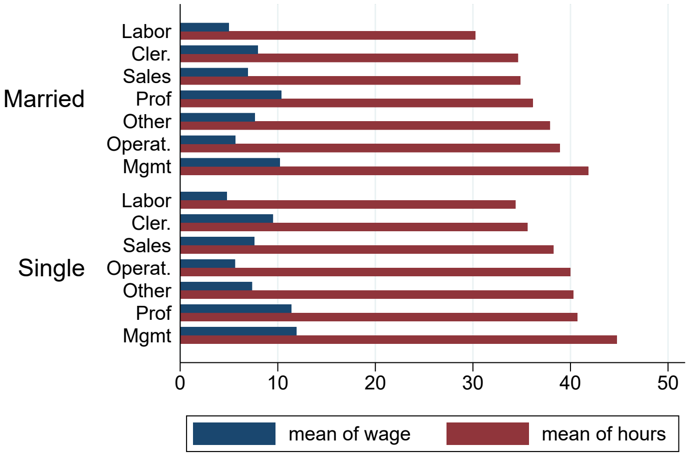
```
graph hbar wage hours, over(occ7, sort(2)) over(married)
```

We can use the `sort()` option when there are multiple `over()` variables. Here the `sort(2)` option orders the bars according to the mean number of hours worked within each level of `married`. *Uses nlsw.dta*



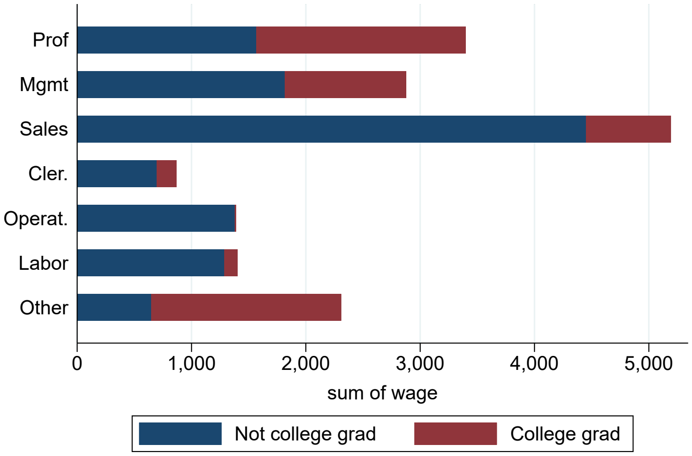
```
graph hbar wage hours, over(occ7, sort(2)) over(married,
descending)
```

Each `over()` option can have its own separate sorting options. In this example, we add the `descending` option to the second `over()` option, and the levels of `married` are now shown with those who are married appearing first. *Uses nlsw.dta*



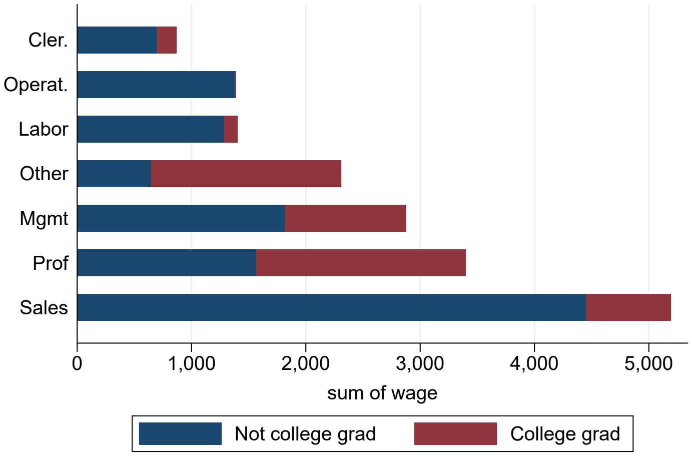
```
graph hbar (sum) wage, over(collgrad) over(occ7) asyvars
stack
```

Say that we were to graph the sum of `wage` broken down by `collgrad` and `occ7`. We further treat the levels of `collgrad` as y variables and form a stacked bar chart. We might want to sort these bars on the sum of wages for each occupation. See the next example for how we can do that. *Uses nlsw.dta*



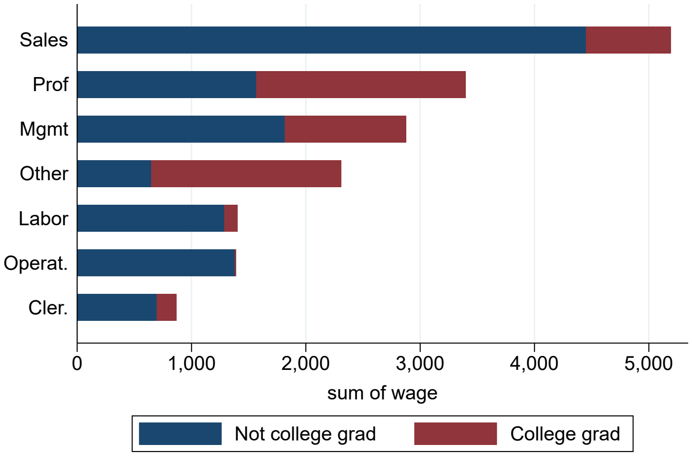
```
graph hbar (sum) wage, over(collgrad) over(occ7, sort((sum)
wage)) asyvars stack
```

Here we add `sort((sum) wage)` to the `over()` option for `occ7`. The bars are sorted on the sum of wages at each level of `occ7`, sorting the bars on their total height. *Uses nlsw.dta*



```
graph hbar (sum) wage, over(collgrad) over(occ7, sort((sum)
wage) descending) asyvars stack
```

In this example, we add the `descending` option to change the sort order from highest to lowest. Note the placement of the `descending` option outside the `sort()` option. *Uses nlsw.dta*


Sales
Prof
Mgmt
Other
Labor
Operat.
Cler.
0
1,000
2,000
3,000
4,000
5,000
sum of wage
Not college grad
College grad

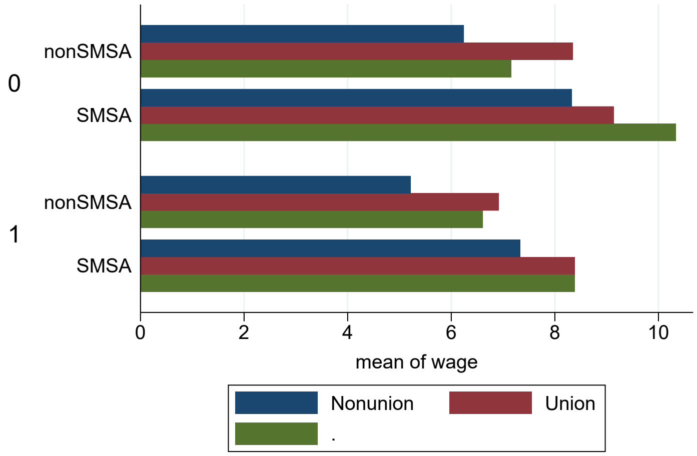
4.5 Controlling the categorical axis

This section describes ways that we can label categorical axes. Bar charts are special because their x axis is formed by categorical variables. This section describes options we can use to customize these categorical axes. For more details, see `help cat_axis_label_options` and `help cat_axis_line_options`.

Let's start by exploring how we can change the labels for the bars on the x axis.

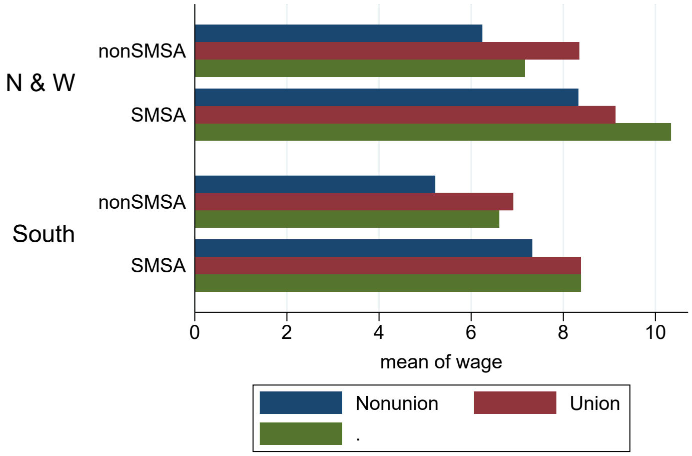
```
graph hbar wage, over(union) over(smsa) over(south) missing
```

This bar chart breaks down wages by union status, whether one lives in the south, and whether one lives in a metropolitan area (`smsa`). Each variable could use a better label to make the graph clearer. The following examples will illustrate how to change the labels for these variables. *Uses nlsw.dta*



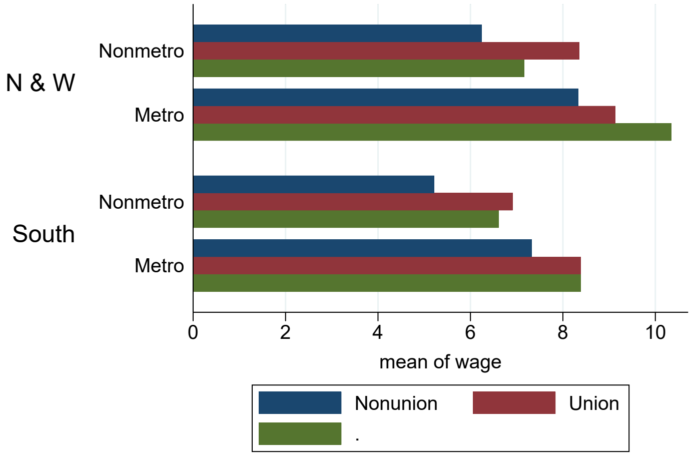
```
graph hbar wage, over(union) over(smsa) over(south, relabel(1
"N & W" 2 "South")) missing
```

The `relabel()` option changes the labels displayed for the levels of `south`. We wrote `relabel(1 "N & W")` and not `relabel(0 "N & W")` because these numbers do not represent the actual levels of `south` but the ordinal position of the levels, that is, first and second. *Uses nlsw.dta*



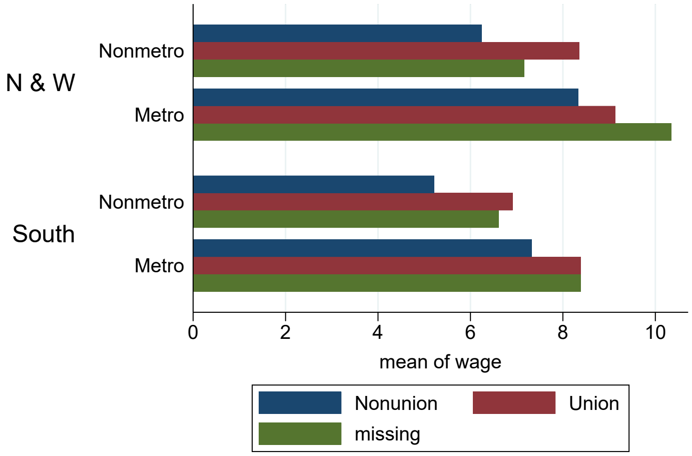
```
graph hbar wage, over(union) over(smsa, relabel(1 "Nonmetro"
2 "Metro")) over(south, relabel(1 "N & W" 2 "South")) missing
```

We now add a `relabel()` option to change the labels for the levels of `smsa`. Each `over()` option can have its own `relabel()` option. *Uses nlsw.dta*



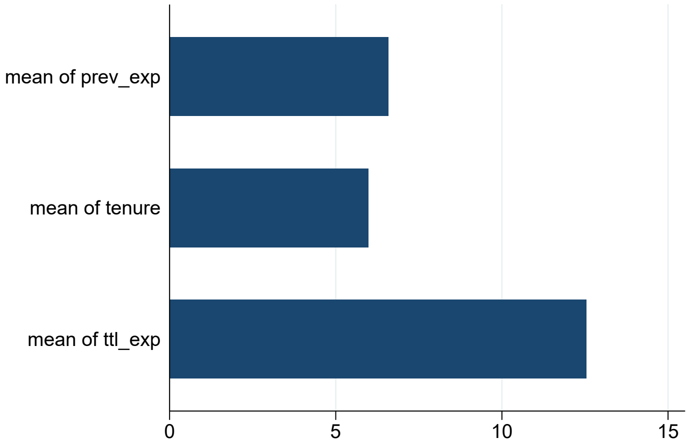
```
graph hbar wage, over(union, relabel(3 "missing")) over(smsa,
relabel(1 "Nonmetro" 2 "Metro")) over(south, relabel(1 "N &
W" 2 "South")) missing
```

We can even use the `relabel()` option on the first `over()` option. Because this variable is labeled with a legend, we could also use the `legend()` option. See Bar : Legend (section 4.6) for more details. *Uses nlsw.dta*



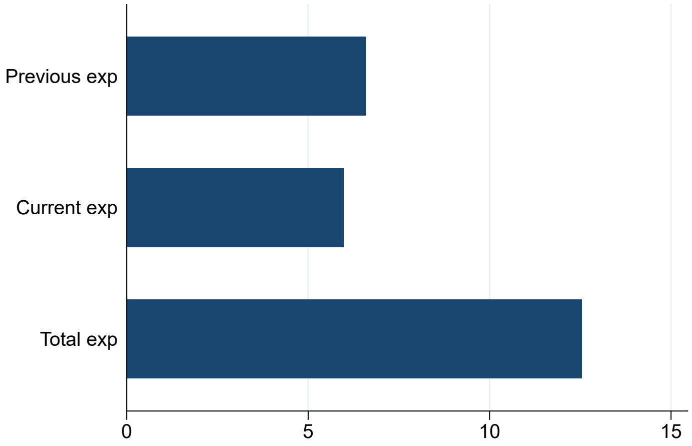
```
graph hbar prev_exp tenure ttl_exp, ascategory
```

This bar chart shows three y variables, but we use the `ascategory` option to plot the different y variables as categorical variables on the x axis. The default labels on the x axis are not bad, but we might want to change them. *Uses nlsw.dta*



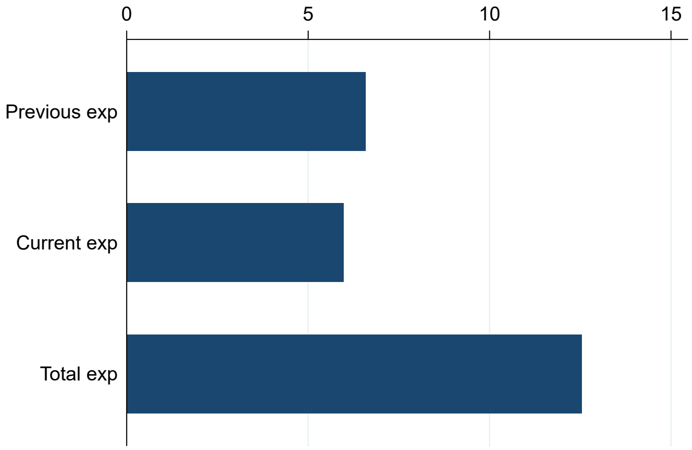
```
graph hbar prev_exp tenure ttl_exp, ascategory
yvaroptions(relabel(1 "Previous exp" 2 "Current exp" 3 "Total
exp"))
```

If the three level-of-experience variables were indicated by an `over()` option, we would use the `over(, relabel())` option to change the labels. Instead, because we have treated the multiple y variables as categories, we then use `yvaroptions(relabel())` to modify the labels on the x axis. *Uses nlsw.dta*



```
graph hbar prev_exp tenure ttl_exp, ascategory yalternate
yvaroptions(relabel(1 "Previous exp" 2 "Current exp" 3 "Total
exp"))
```

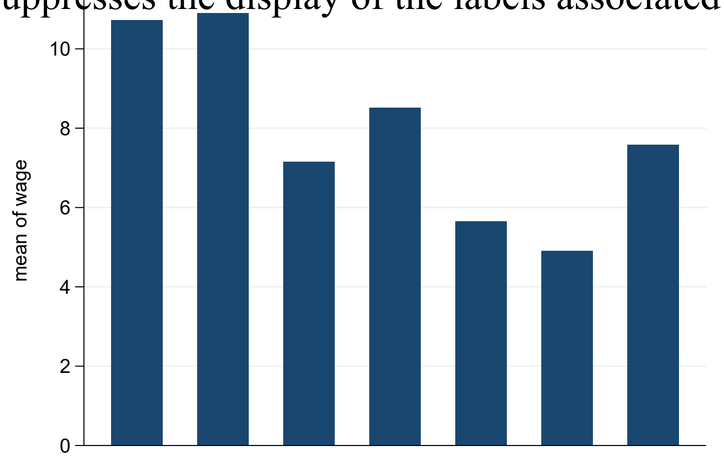
We move the y axis to the opposite side of the graph by using the `yalternate` option. You can also use the `xalternate` option to move the x axis to its opposite side. *Uses nlsw.dta*



Previously, we saw that we can use the `relabel()` option within the `over()` option to control the labeling of `over()` variables and within `yvaroptions()` to control the labeling of multiple y variables (provided that we use the `ascategory` option to convert the multiple y variables into categories). Let's now explore other `over()` options, which we can use with either `over()` or `yvaroptions()`.

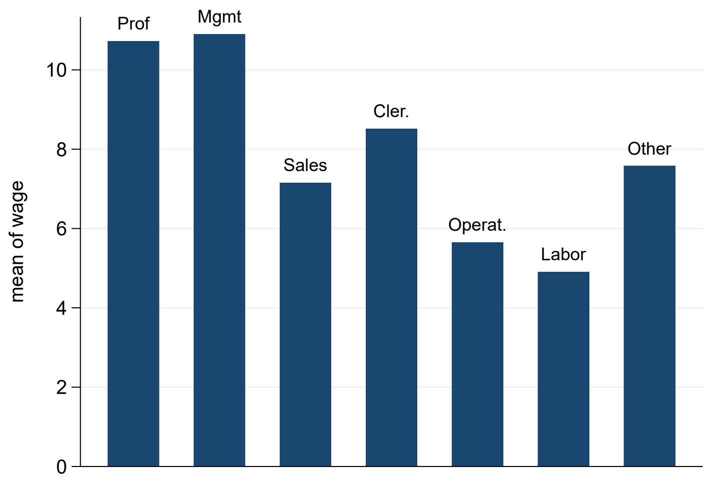
```
graph bar wage, over(occ7, label(nolabels))
```

The `label(nolabels)` option suppresses the display of the labels associated with the levels of `occ7`. The `label(nolabels)` option is generally not useful alone but is very useful in combination with other means to label the bars. Consider the next example. *Uses nlsw.dta*



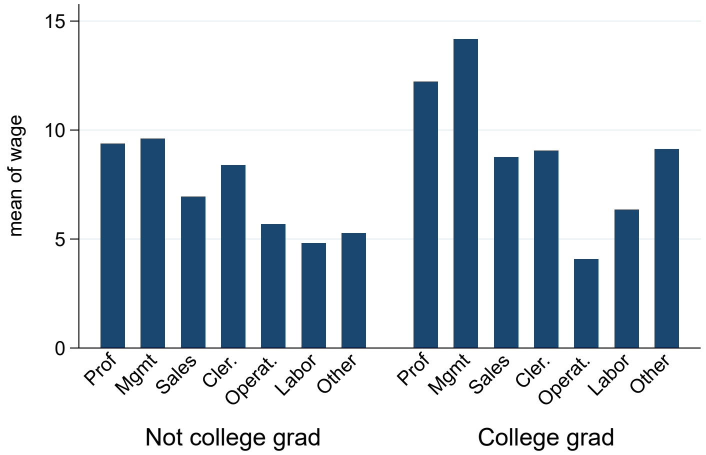
```
graph bar wage, over(occ7, label(nolabels)) blabel(group)
```

By adding the `blabel(group)` (bar label) option, we label the bars with the name of the group to which the bar belongs. See Bar : Legend (section 4.6) for more about `blabel()`. *Uses nlsw.dta*



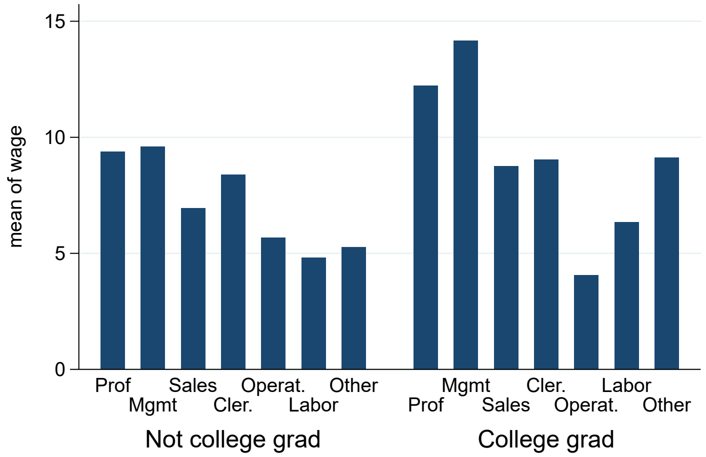
```
graph bar wage, over(occ7, label(angle(45))) over(collgrad)
```

This graph shows wages broken down by occupation and by whether one graduated college. We add the `label(angle(45))` option to rotate the labels for occupation by 45 degrees. If this had been omitted, the labels would have overlapped each other. *Uses nlsw.dta*



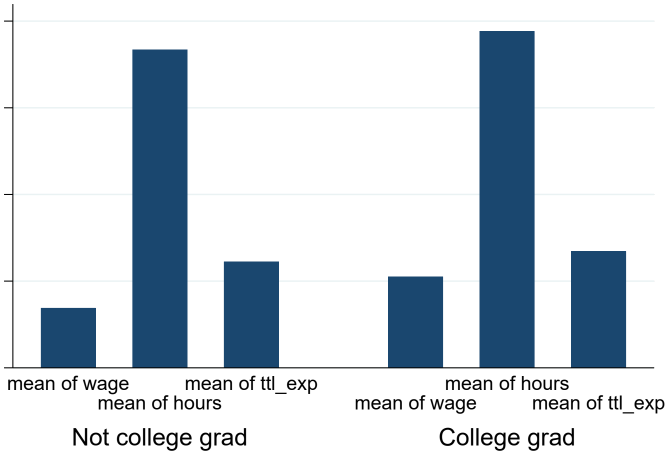
```
graph bar wage, over(occ7, label(alternate)) over(collgrad)
```

Compare this graph with the previous example. This example uses the `label(alternate)` option to avoid overlapping by alternating the labels for occupation. *Uses nlsw.dta*



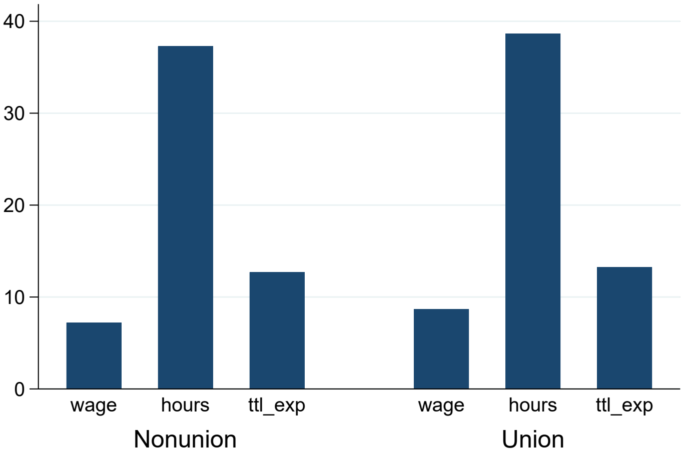
```
graph bar wage hours ttl_exp, ascategory over(collgrad)
yvaroptions(label(alternate))
```

This example illustrates using the `label(alternate)` option to alternate labels created by multiple y variables converted to categories using the `ascategory` option. Then the option is specified as `yvaroptions(label(alternate))`. *Uses nlsw.dta*



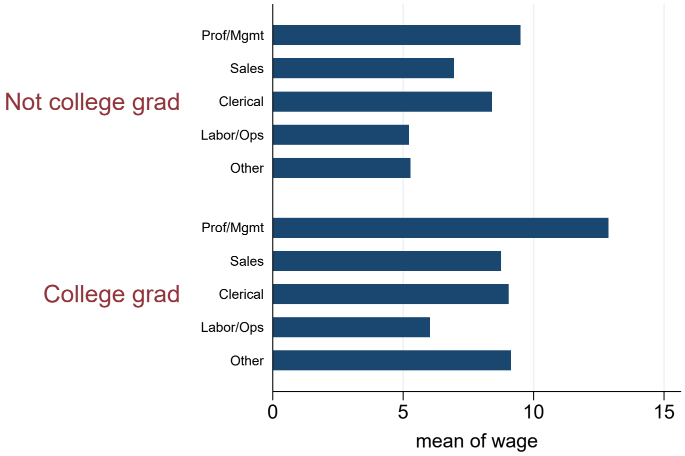
```
graph bar wage hours ttl_exp, ascategory over(union) nolabel
```

If we add the `nolabel` option, the names of the variables are shown instead of the value labels. *Uses nlsw.dta*



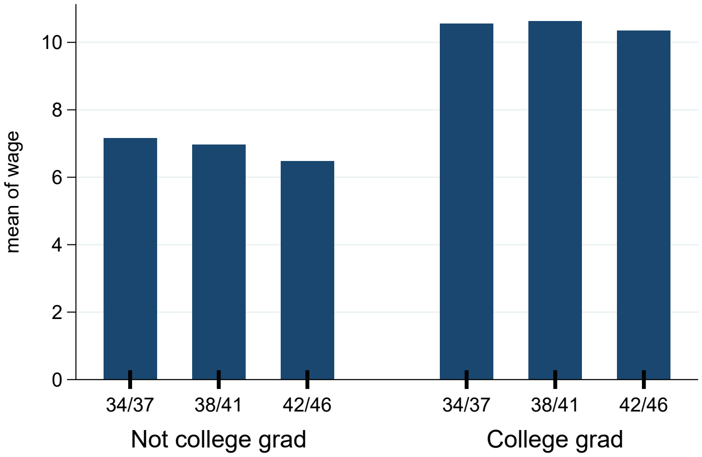
```
graph hbar wage, over(occ5, label(labsize(small)))
over(collgrad, label(labcolor(maroon)))
```

Here we make the labels for `occ5` small and the color of the labels for `collgrad` maroon. See **Styles : Colors** (section 10.2) for other color options and **Styles : Textsize** (section 10.12) for size options (including sizing using points, inches, and centimeters). *Uses nlsw.dta*



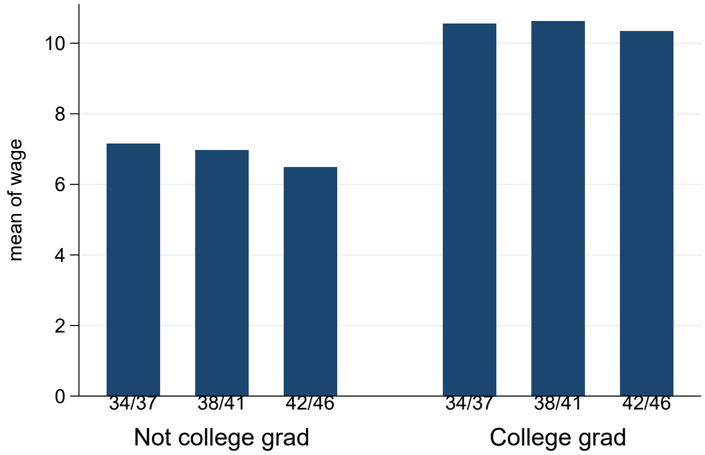
```
graph bar wage, over(age3, label(ticks tlwidth(thick)
tlength(medium) tposition(crossing))) over(collgrad)
```

The `ticks` option adds tick marks. Here we make them thick in width and medium in length, and they cross the x axis. See `help cat_axis_label_options` for more details. *Uses nlsw.dta*



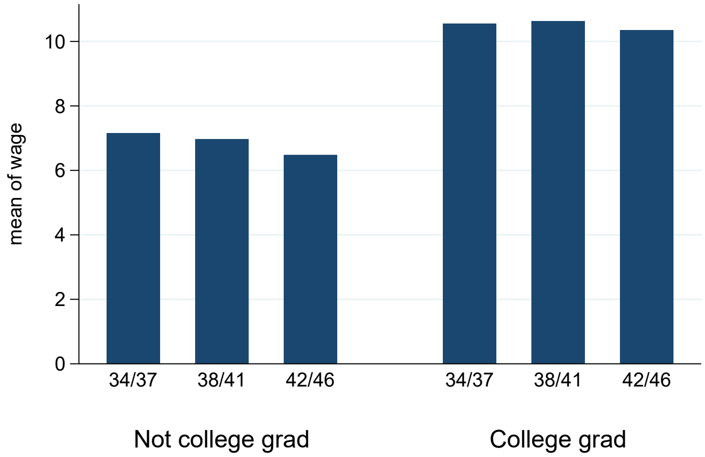
```
graph bar wage, over(age3, label(labgap(zero)))
over(collgrad)
```

We use the `labgap(zero)` option to make no gap between the label and the axis. *Uses nlsw.dta*



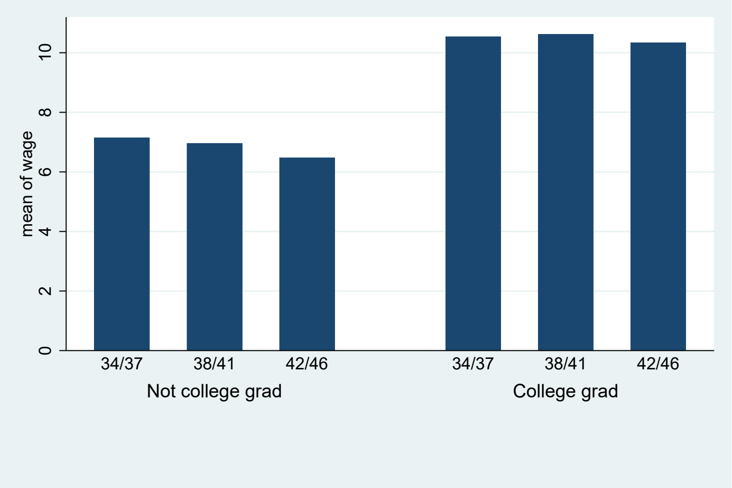
```
graph bar wage, over(age3) over(collgrad,
label(labgap(huge)))
```

We use the `label(labgap(huge))` option to control the gap between the labels for `age3` and `collgrad`, making that gap huge. *Uses nlsw.dta*



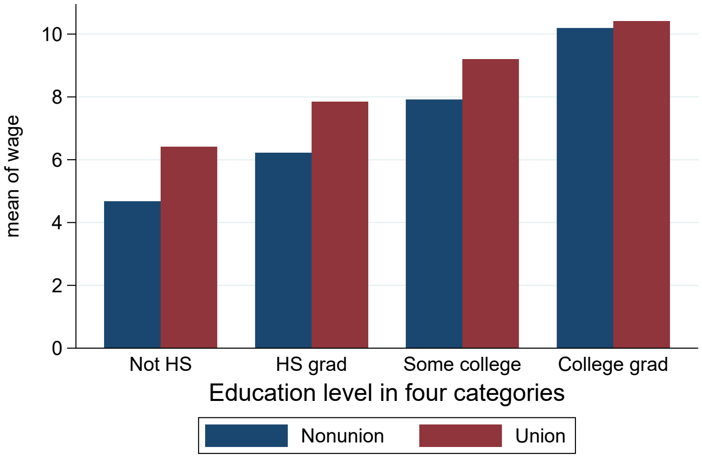
```
graph bar wage, over(age3) over(collgrad,
axis(outergap(*20))) scheme(s2color)
```

The `axis(outergap(*20))` option controls the gap between the labels of the x axis and the outside of the graph. To help make this gap more noticeable, I included the `scheme(s2color)` option for this graph. As you can see, including this option created considerable space below the labels for `collgrad` and the bottom of the graph. *Uses nlsw.dta*



```
graph bar wage, over(union) over(grade4) asyvars
b1title("Education level in four categories")
```

The `b1title()` option adds a title to the bottom of the graph, in effect labeling the x axis. We can add a second title below that by using the `b2title()` option. If you used `graph hbar`, you could label the left axis using the `l1title()` and `l2title()` options. *Uses nlsw.dta*

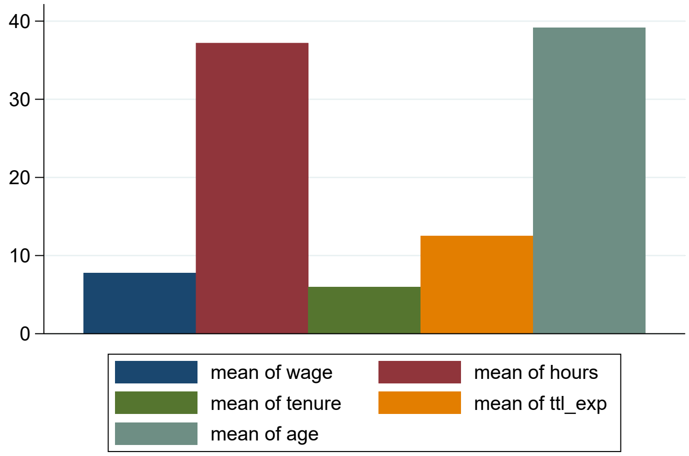


4.6 Controlling the legend and labeling bars

This section discusses the use of the legend for bar charts, emphasizing the features that are unique to bar charts. The section **Options : Legend** (section 8.9) goes into great detail about the legend, as does `help legend_options`. You can use the legend for multiple y variables or when the first `over()` variable is treated as a y variable with the `asyvars` option. See **Bar : Y-variables** (section 4.1) for more information about the use of multiple y variables and **Bar : Over** (section 4.2) for more examples of treating the first `over()` variable as a y variable. The following examples show the different kinds of labels that we can create by using the `blabel()` option. We can create labels that display the name of y variable, the name of the first `over()` group, the height of the bar, or the overall height of the bar (when used with the `stack` option).

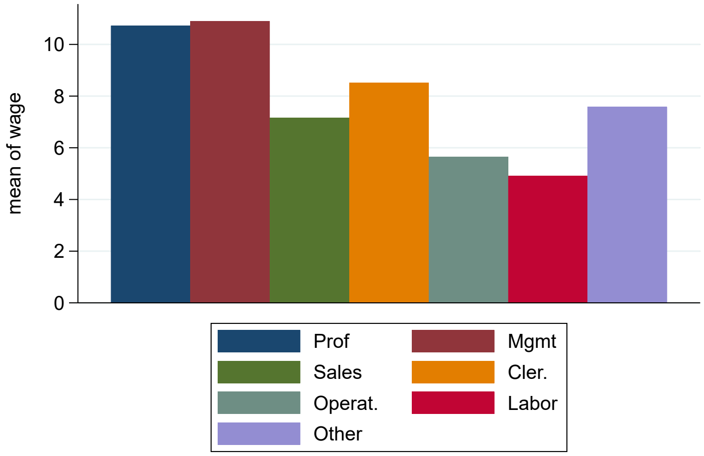
```
graph bar wage hours tenure ttl_exp age
```

Consider this bar graph of five different y variables. The bars for the different y variables are shown with different colors, and a legend is used to identify the y variables. *Uses nlsw.dta*



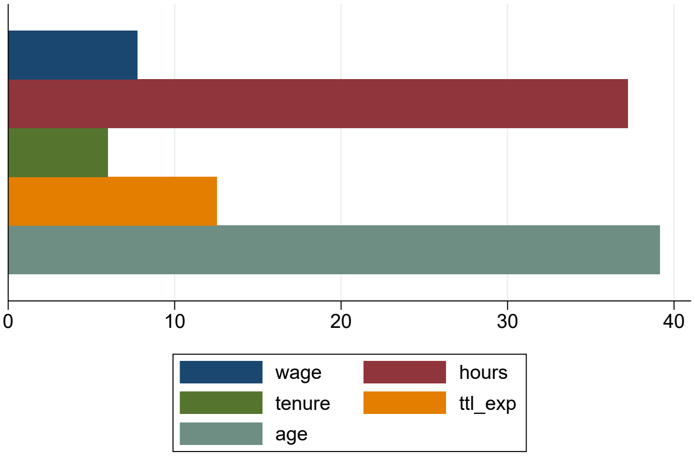
```
graph bar wage, over(occ7) asyvars
```

This is another example of where a legend can arise in a Stata bar graph by specifying the `asyvars` option, which treats an `over()` variable as though the levels were different y variables. *Uses nlsw.dta*



Unless otherwise mentioned, the legend options described below work the same regardless of whether we derived the legend from multiple y variables or from an `over()` variable that was combined with the `asyvars` option.

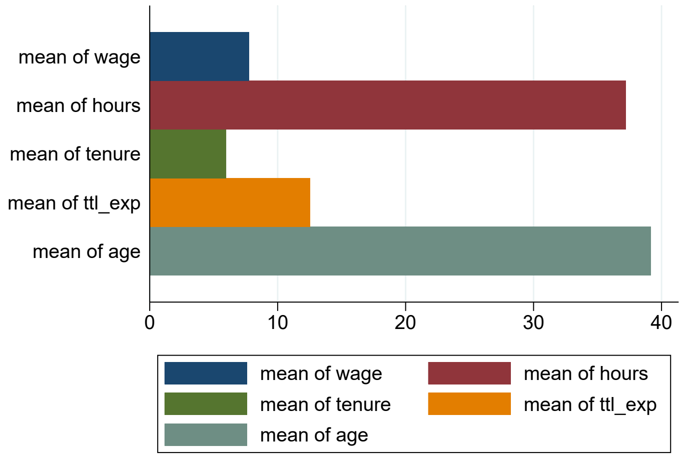
```
graph hbar wage hours tenure ttl_exp age, nolabel
```



The `nolabel` option only works when we have multiple y variables. When it is used, the variable names (not the variable labels) are used in the legend. For example, instead of showing the variable label `hourly wage`, it shows the variable name `wage`. *Uses nlsw.dta*

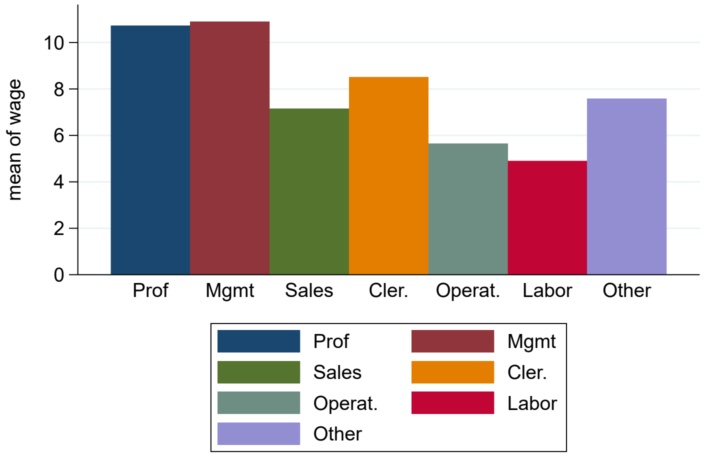
```
graph hbar wage hours tenure ttl_exp age, showyvars
```

The `showyvars` option puts the labels on the axis, beside or “under” the bars. *Uses nlsw.dta*



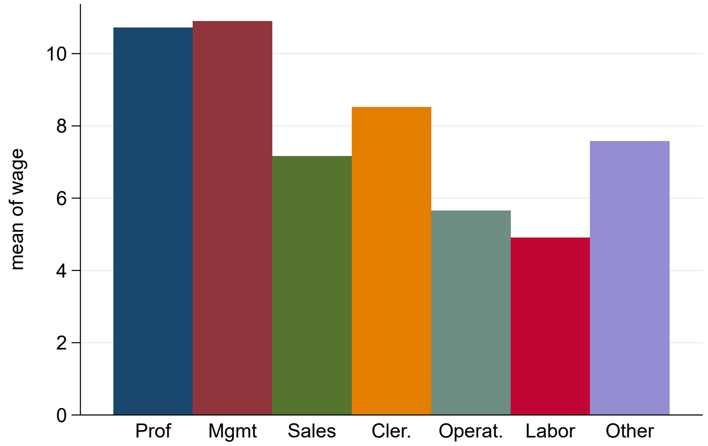
```
graph bar wage, over(occ7) asyvars showyvars
```

Even though the `showyvars` option sounds like it would work only with multiple y variables, it also works when we combine the `over()` and `asyvars` options. As we can see, the legend is now redundant and could be suppressed. *Uses nlsw.dta*



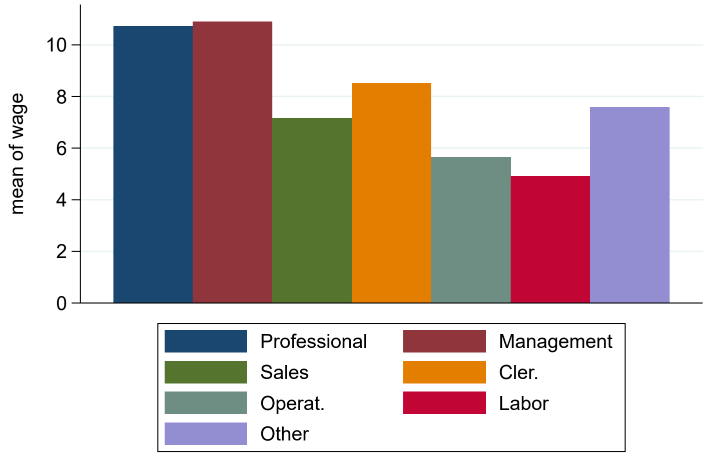
```
graph bar wage, over(occ7) asyvars showyvars legend(off)
```

This example is similar to the previous one, but here we use the `legend(off)` option to suppress the display of the legend. *Uses nlsw.dta*



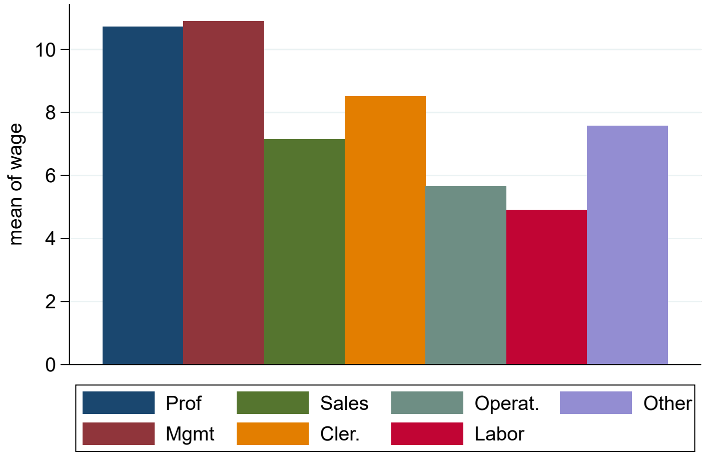
```
graph bar wage, over(occ7) asyvars legend(label(1
"Professional") label(2 "Management"))
```

We use the `legend(label())` option to change the labels for the first two bars in the graph. We use a separate `label()` option for each bar. *Uses nlsw.dta*



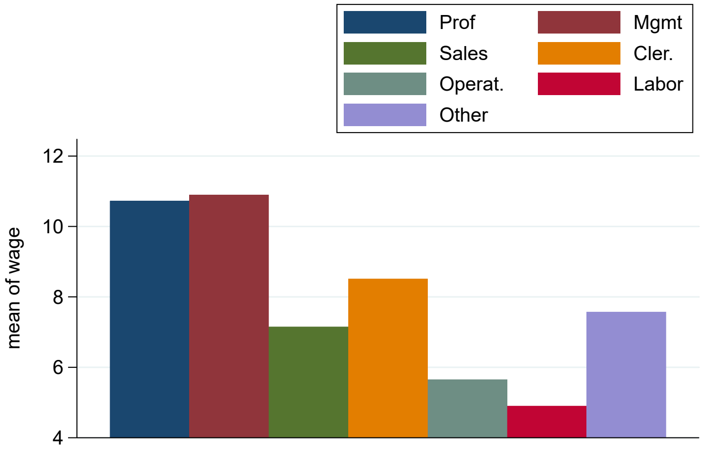
```
graph bar wage, over(occ7) asyvars legend(rows(2) colfirst)
```

The `rows(2)` option combined with `colfirst` displays the legend in two rows with the keys ordered by column (instead of by row, the default). This yields keys that are more adjacent to the bars that they label. *Uses nlsw.dta*



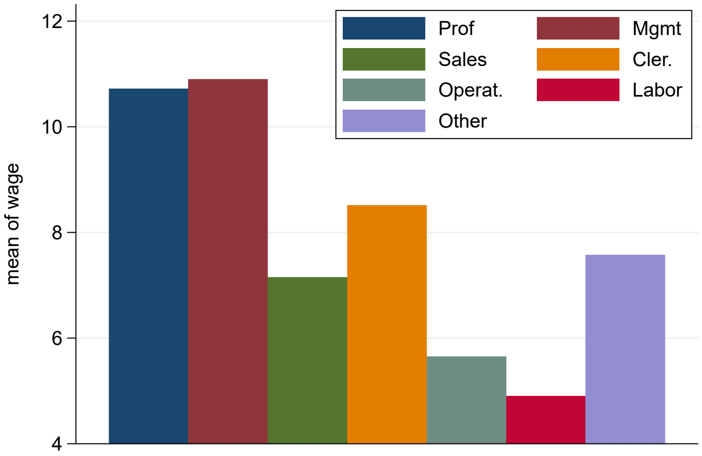
As we can see, the default placement for the legend is below the x axis. However, Stata gives us flexibility in the placement of the legend. Let's now consider options that control the placement of the legend, along with options useful for controlling the placement of the items within the legend.

```
graph bar wage, over(occ7) asyvars exclude0
legend(position(1))
```



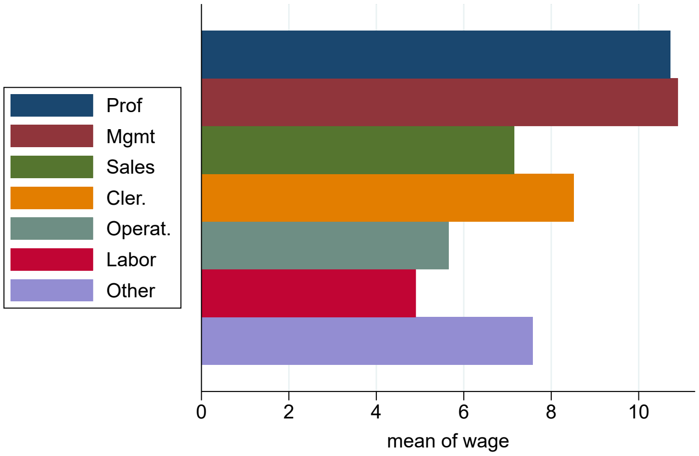
We can use the `legend(position(1))` option to place the legend in the top right corner of the graph. The values supplied for `position()` are like the numbers on a clock face, where twelve o'clock is the top, six o'clock is the bottom, and 0 represents the center of the clock face. Specifying one o'clock places the legend in the top right; see Styles : Clockpos (section 10.3) for more details. *Uses nlsw.dta*

```
graph bar wage, over(occ7) asyvars exclude0
legend(position(1) ring(0))
```



By adding the `ring(0)` option, we can tuck the legend inside the top right corner of the plot area. Think of the `ring()` option as specifying concentric rings around the graph, where 0 is a position inside the plot region, 1 is just outside the plot region, and increasing values are farther from the center of the plot region. *Uses nlsw.dta*

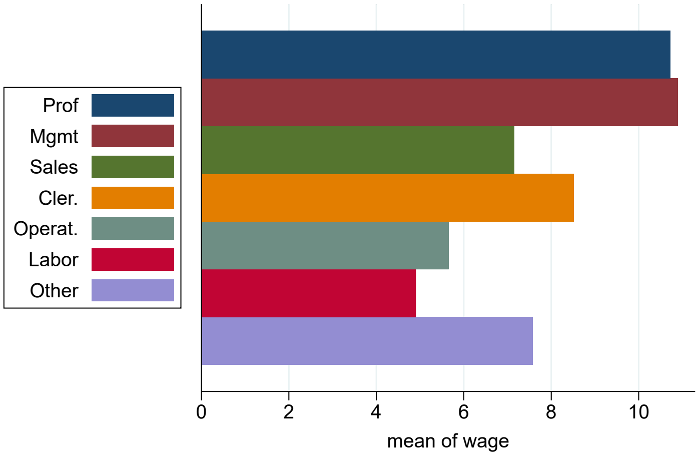
```
graph hbar wage, over(occ7) asyvars legend(cols(1)
position(9))
```



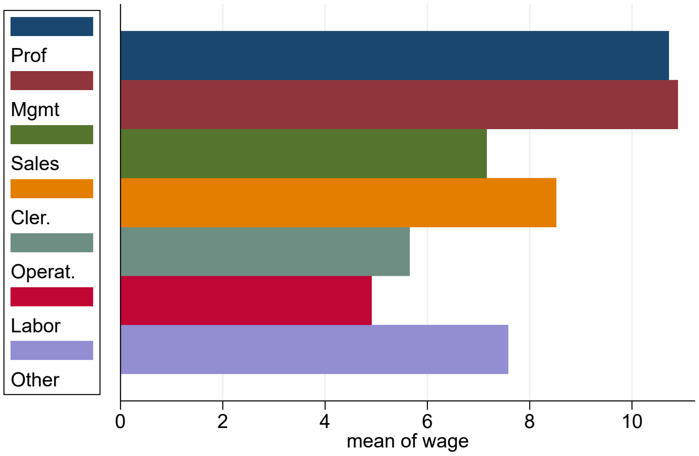
By using `graph hbar`, we switch this to a horizontal bar chart. The legend is placed in the nine o'clock position with the `position(9)` option and displays in one column with the `cols(1)` option. *Uses nlsw.dta*

```
graph hbar wage, over(occ7) asyvars legend(cols(1)
position(9) textfirst)
```

Adding the `textfirst` option places the description of the key before the symbol in the legend. *Uses nlsw.dta*



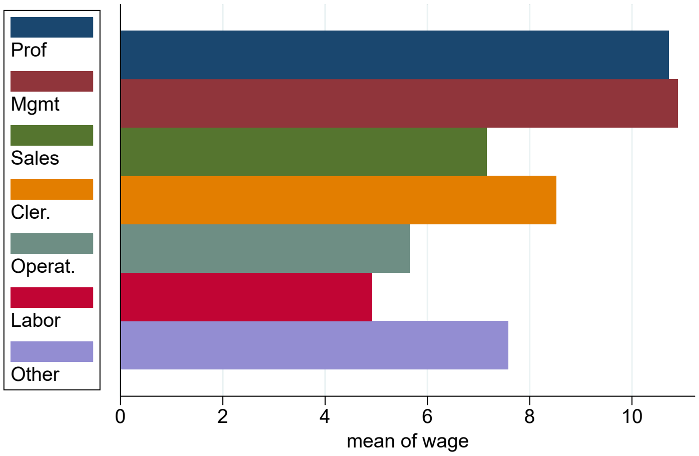
```
graph hbar wage, over(occ7) asyvars legend(cols(1)
position(9) stack)
```



The `stack` option stacks the keys and their labels. Other options for controlling the legend include `rowgap()`, `keygap()`, `symxsize()`, `symysize()`, `textwidth()`, and `symplacement()`. See Options : Legend (section 8.9) and `help legend_options` for more details. *Uses nlsw.dta*

```
graph hbar wage, over(occ7) asyvars scheme(vg_lgndc)
```

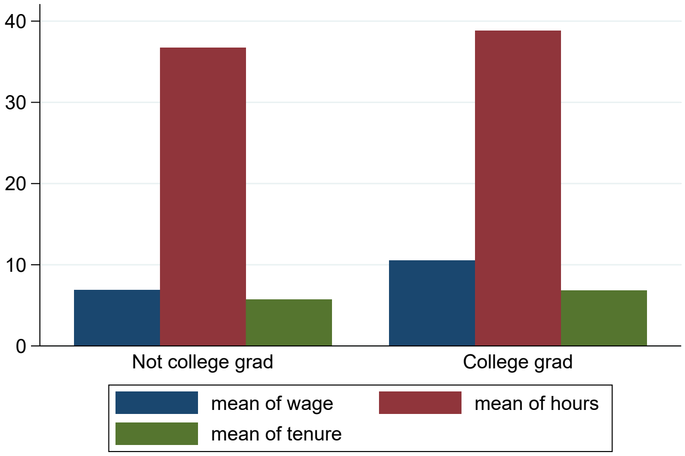
This example uses the `scheme(vg_lgndc)` option. Notice how it positions and customizes the legend, as in the previous example. With this scheme, the legend defaults to the nine o'clock position, in a single column, with the keys and symbols stacked. *Uses nlsw.dta*



Now let’s look at how we can use the `blabel()` (bar label) option to add labels to the bars. These labels can show the name of the `over()` option, the name of y variables, or the height of the bar. These options are illustrated below along with other related options we might use in conjunction with `blabel()` for identifying the bars.

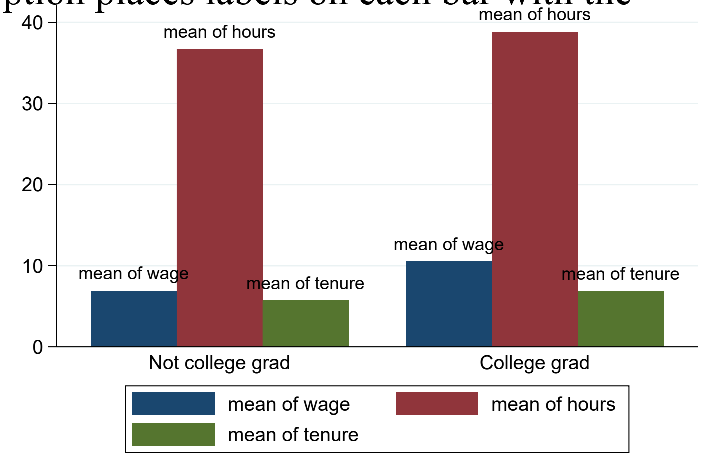
```
graph bar wage hours tenure, over(collgrad)
```

Consider this graph, where we look at `wage`, `hours`, and `tenure` broken down by the levels of `collgrad`. The legend identifies the bars for us. There are ways, in addition to the legend, to label these bars, as we will see in the upcoming examples. *Uses nlsw.dta*



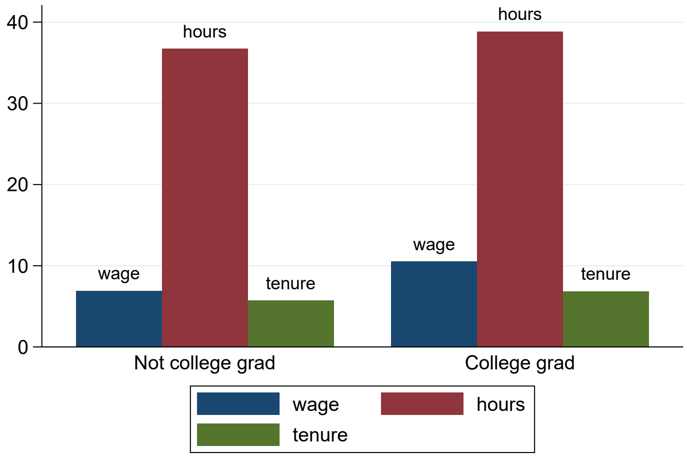
```
graph bar wage hours tenure, over(collgrad) blabel(name)
```

The `blabel(name)` (bar label) option places labels on each bar with the name of the y variable. Here each label is preceded with *mean of* because each bar represents the mean of y variable. *Uses nlsw.dta*



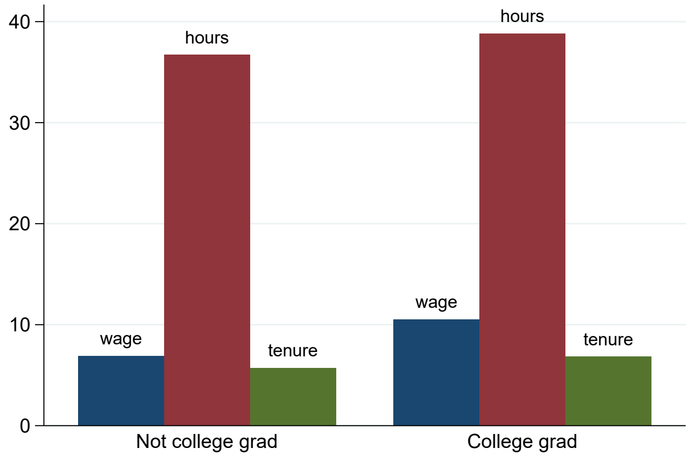
```
graph bar wage hours tenure, over(collgrad) blabel(name)
nolabel
```

The `nolabel` option specifies to show only the name of the y variable. For example, instead of showing the label *mean of wage*, it shows the variable name `wage`. *Uses nlsw.dta*



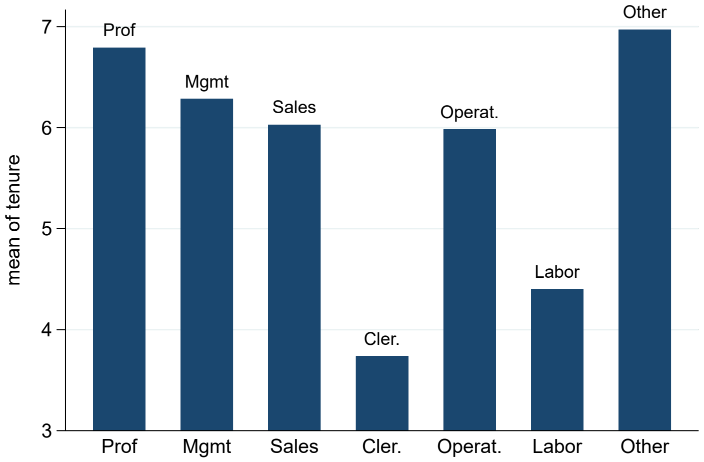
```
graph bar wage hours tenure, over(collgrad) blabel(name)
nolabel legend(off)
```

Now the legend is no longer needed, so we can suppress the display of the legend with the `legend(off)` option. See Options : Legend (section 8.9) for more information about legend options. *Uses nlsw.dta*



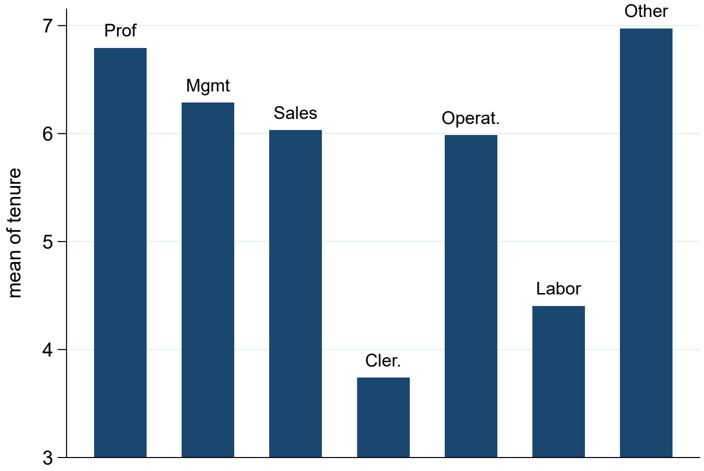
```
graph bar tenure, over(occ7) exclude0 blabel(group)
```

The `blabel(group)` option shows the label for the first `over()` group at the top of each bar. Here the label at the bottom of the bar becomes unnecessary. *Uses nlsw.dta*

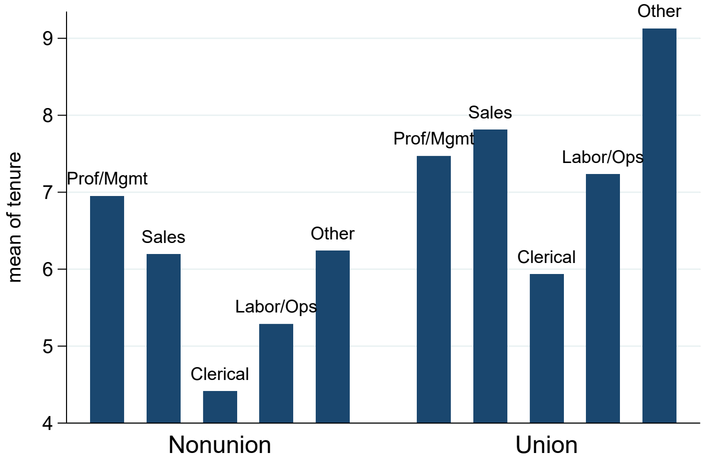


```
graph bar tenure, over(occ7, label(nolabels)) exclude0
blabel(group)
```

The `label(nolabels)` option suppresses the display of the labels below each bar. *Uses nlsw.dta*



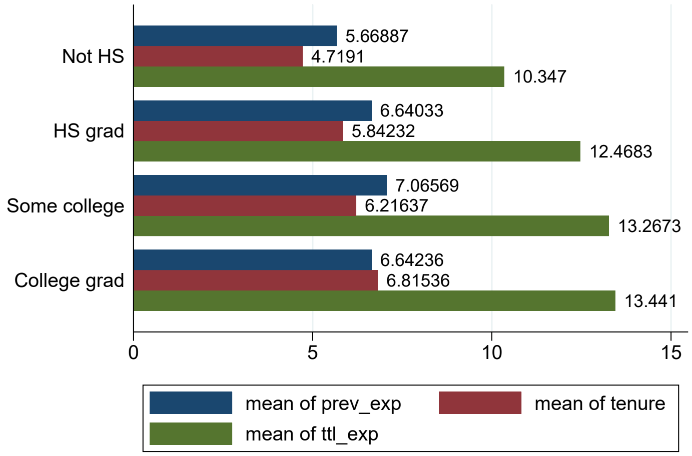
```
graph bar tenure, over(occ5, label(nolabels)) exclude0
blabel(group) yscale(range(7.2)) over(union)
```



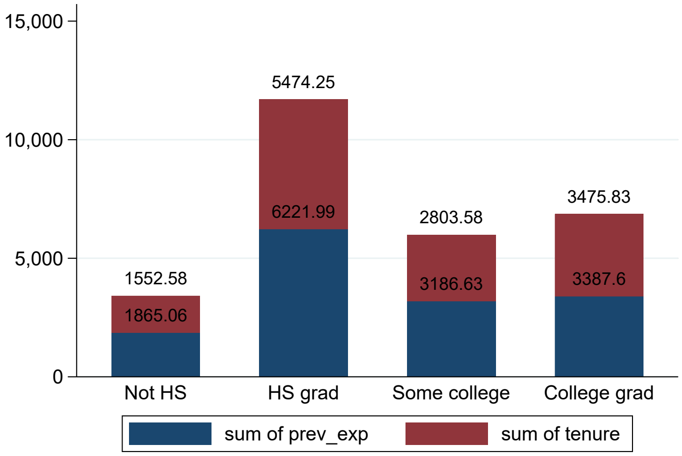
Even if we add a second `over()` option, the levels of the first `over()` variable are labeled at the top of each bar because of the `blabel()` option, and the levels of the second `over()` variable are labeled, as usual, at the bottom of the bars. The `blabel()` option does not work this way when you have three `over()` options or multiple y variables. *Uses nlsw.dta*

```
graph hbar prev_exp tenure ttl_exp, over(grade4) blabel(bar)
```

Consider this graph showing previous, current, and total work experience broken down by education. Here we use the `blabel(bar)` option to display the bar height (that is, the mean of y variables). *Uses nlsw.dta*



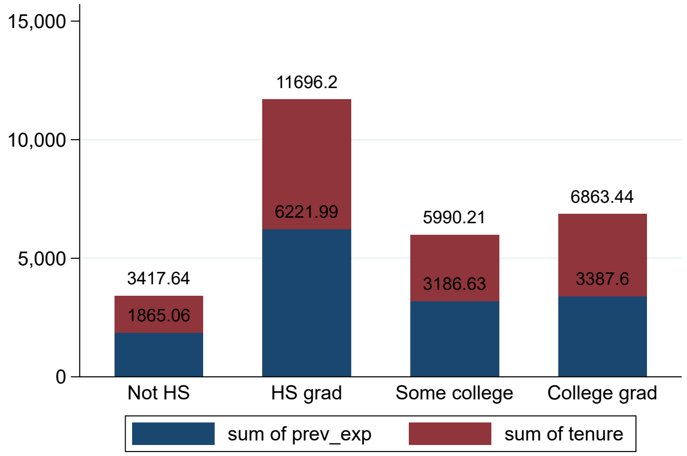
```
graph bar (sum) prev_exp tenure, stack over(grade4)
blabel(bar)
```



Using the `(sum)` statistic, this graph shows the total experience for all individuals in a grade level before their current job (`prev_exp`) and the sum of experience for all individuals in a grade level in their current job (`tenure`) and then uses `stack` to stack these two totals. With the `blabel(bar)` option, the bar labels are the totals for each y variable broken down by `grade4`. *Uses nlsw.dta*

```
graph bar (sum) prev_exp tenure, stack over(grade4)
blabel(total)
```

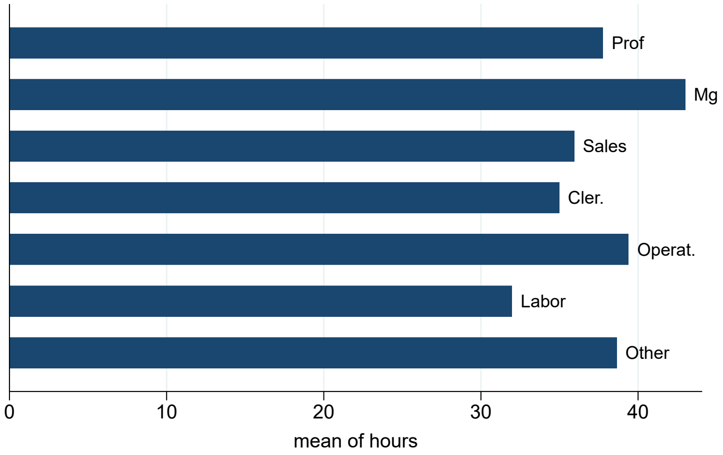
As compared with the prior example, this example uses the `blabel(total)` option to display the results as totals. Now the labels represent the cumulative total height of the bar. *Uses nlsw.dta*



We have seen several ways that we can use the `blabel()` option to label the bars. Stata also offers other options to control the display of these labels. Next let’s consider some of these options to customize the way these labels are displayed.

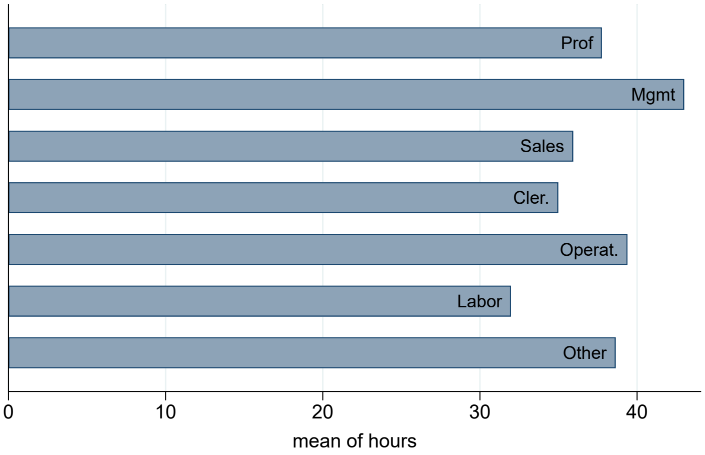
```
graph hbar hours, over(occ7, label(nolabels)) blabel(group)
```

Consider this graph of the average hours worked by occupation. We add labels of the occupation at the top of each bar but suppress the label at the bottom of each bar. Note how the label for the second bar runs off the right of the graph. *Uses nlsw.dta*



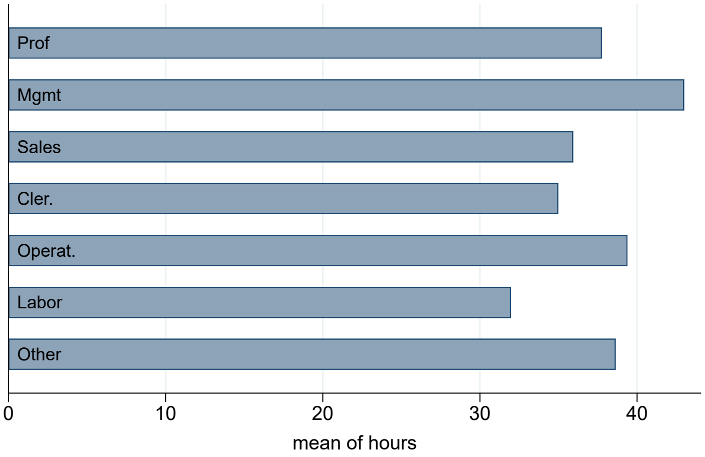
```
graph hbar hours, over(occ7, label(nolabels)) blabel(group,
position(inside)) scheme(vg_palec)
```

The `position(inside)` option places the group label inside the bar. By default, `inside` refers to the “top” inside the bar. Because we added the `scheme(vg_palec)` option, the bar colors are pale, so the labels within the bars are readable. *Uses nlsw.dta*



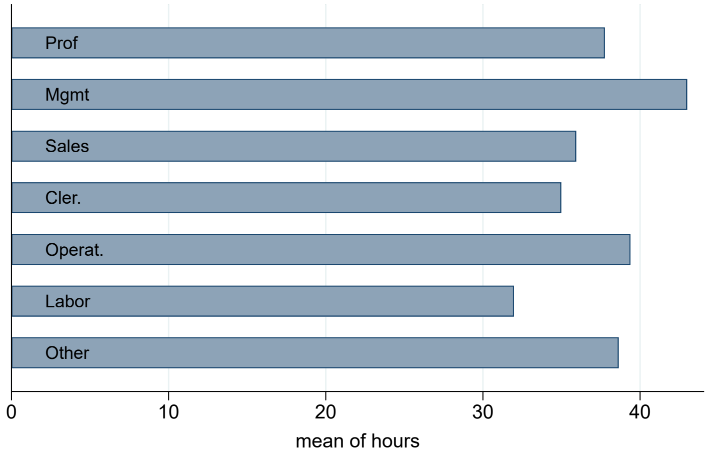
```
graph hbar hours, over(occ7, label(nolabels)) blabel(group,
position(base)) scheme(vg_palec)
```

The `position(base)` option places the label at the “base” inside of the bar. You can also specify `position(center)` to place the label in the center of the bar. This example also uses the `scheme(vg_palec)` option. *Uses nlsw.dta*



```
graph hbar hours, over(occ7, label(nolabels)) blabel(group,
position(base) gap(huge)) scheme(vg_palec)
```

We can use the `gap()` option to fine-tune the placement of the label. Here we position the label at the base but make the gap between the label and the base huge. We can also use the `gap()` option with `position(inside)` to position the label with respect to the top of the bar. *Uses nlsw.dta*

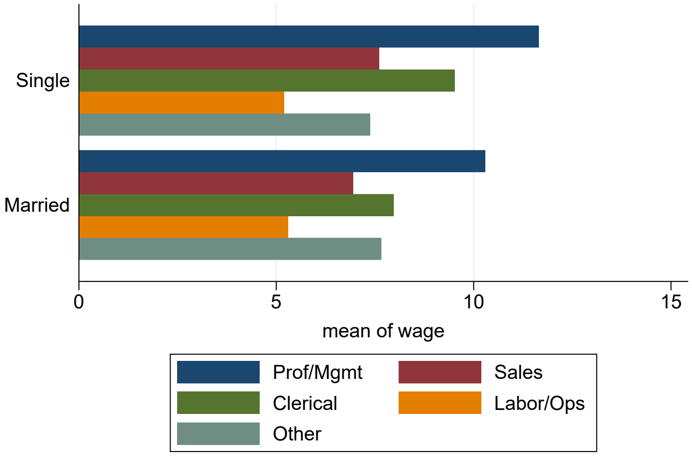

Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other
0
10
20
30
40
mean of hours

4.7 Controlling the y axis

This section describes options to control the y axis in bar charts. To be precise, when Stata refers to the y axis on a bar chart, it refers to the axis with the continuous variable, whether that be the left axis when using `graph bar` or the bottom axis when using `graph hbar`. This section emphasizes the features that are particularly relevant to bar charts. For more details, see Options : Axis titles (section 8.4), Options : Axis labels (section 8.5), and Options : Axis scales (section 8.6). Also see `help axis_title_options`, `help axis_label_options`, and `help axis_scale_options`.

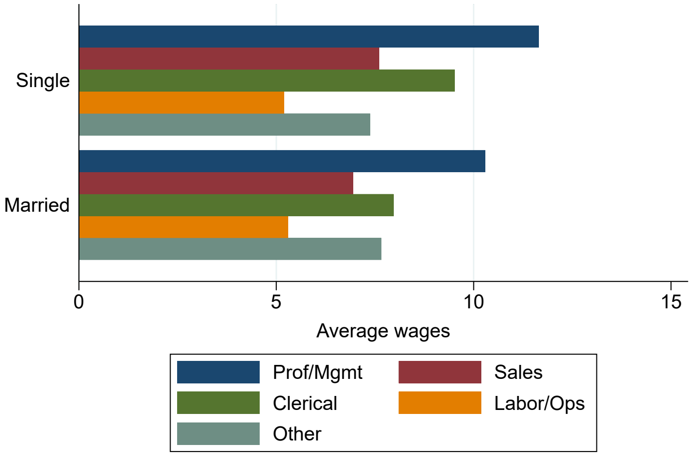
```
graph hbar wage, over(occ5) over(married) asyvar
```

Consider this graph showing the mean hourly wage broken down by occupation and marital status. *Uses nlsw.dta*

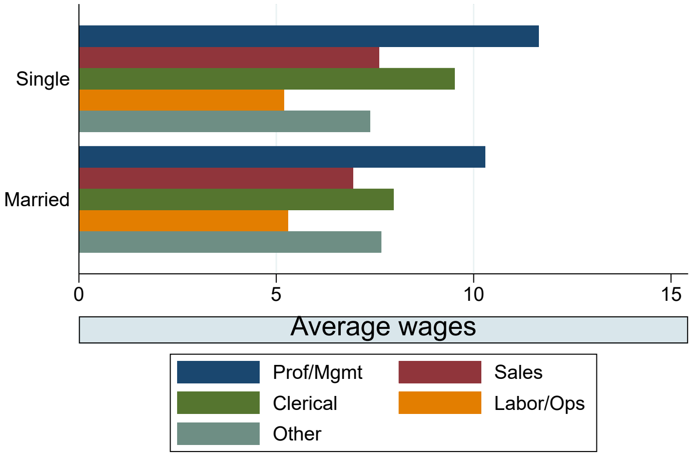


```
graph hbar wage, over(occ5) over(married) asyvar
ytitle("Average wages")
```

We can use the `ytitle()` option to add a title to the y axis. See Options : Axis titles (section 8.4) and `help axis_title_options` for more details, but disregard any references to `xtitle()` because that option is not valid when using `graph bar`. *Uses nlsw.dta*



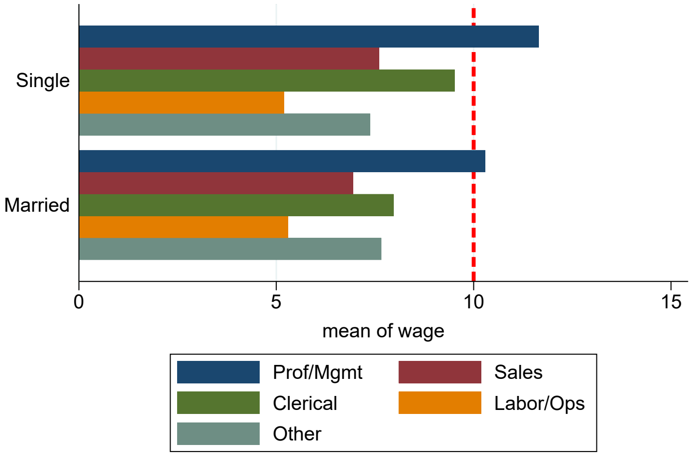
```
graph hbar wage, over(occ5) over(married) asyvar
ytitle("Average wages", size(vlarge) box bexpand)
```



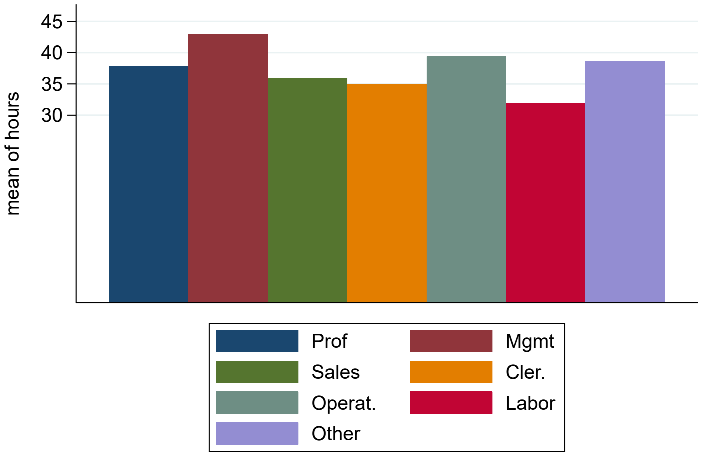
Because this title is considered a textbox, we can use a variety of textbox options to control the look of the title. Here the title is made very large with a box around it; the box is expanded (`bexpand`) to fill the width of the plot area. See Options : Textboxes (section 8.11) for more examples of how to use textbox options to control the display of text (including how to size text in points, inches, and centimeters). *Uses nlsw.dta*

```
graph hbar wage, over(occ5) over(married) asyvar yline(10,
lwidth(thick) lcolor(red) lpattern(dash))
```

Here the `yline()` option is used to place a thick, red, dashed line on the graph where y equals 10. This option is still called `yline()` because the y axis is the axis with the continuous variable. *Uses nlsw.dta*



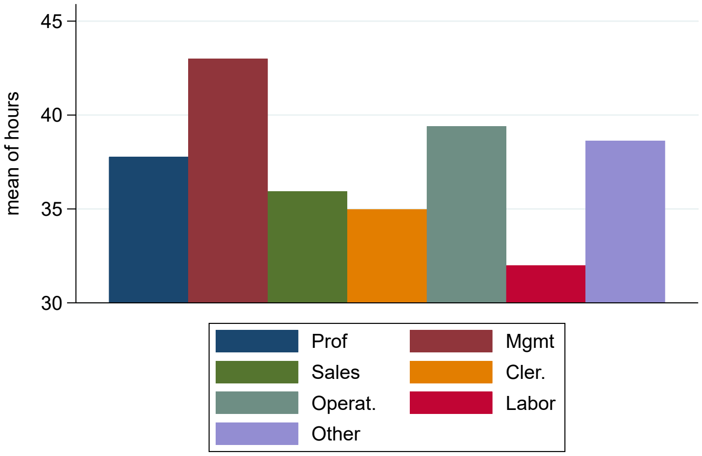
```
graph bar hours, over(occ7) asyvar ylabel(30(5)45)
```



We use the `ylabel()` option to label the y axis from 30 to 45, incrementing by 5. See Options : Axis labels (section 8.5) and `help axis_label_options` for more details. The y axis still begins at 0. See the following example to see how this is controlled. *Uses nlsw.dta*

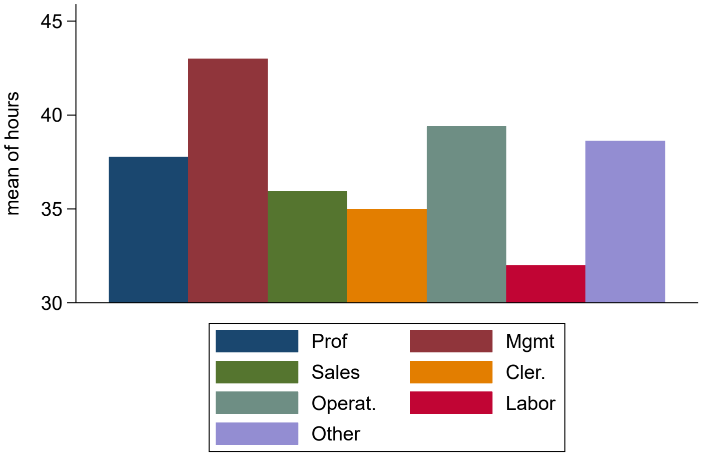
```
graph bar hours, over(occ7) asyvar ylabel(30(5)45) exclude0
```

By default, bar charts include 0 on the y axis. The `exclude0` option removes the 0, as shown here. *Uses nlsw.dta*



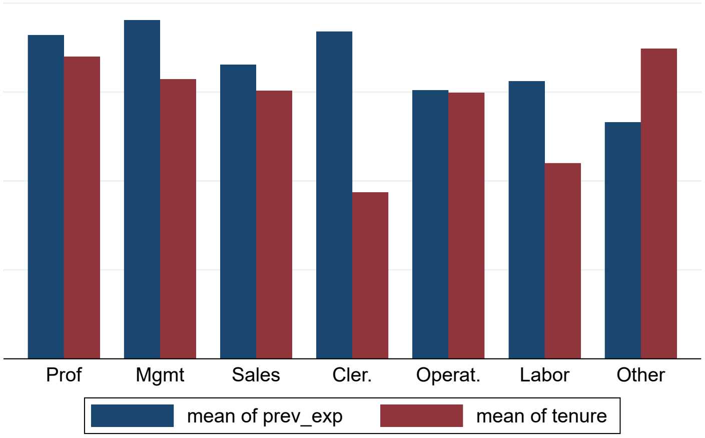
```
graph bar hours, over(occ7) asyvar ylabel(30(5)45, nogrid)
exclude0
```

The `nogrid` option placed within the `ylabel()` option suppresses the grid for the y axis. (With bar charts, there is never a grid with respect to the x axis.) If the grid were absent and you wanted to include it, you could add the `grid` option. For more details, see Options : Axis labels (section 8.5). *Uses nlsw.dta*



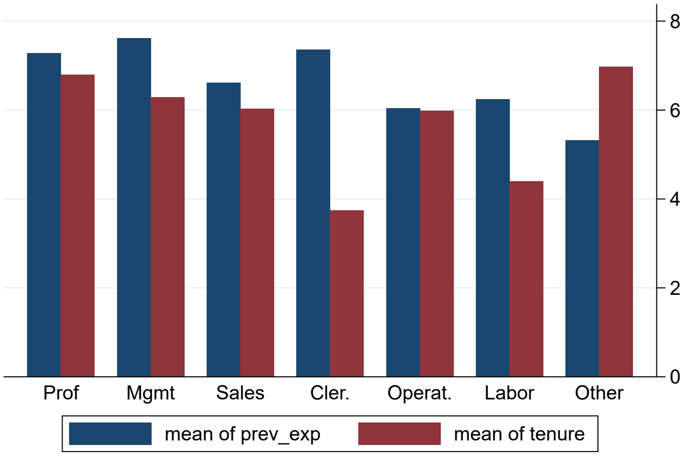
```
graph bar prev_exp tenure, over(occ7) yscale(off)
```

If we want to suppress the display of the y axis entirely, we can use the `yscale(off)` option. See Options : Axis scales (section 8.6) and `help axis_scale_options` for more details. Disregard any references to `xscale()` because that option is not valid when using `graph bar` or `graph hbar`. *Uses nlsw.dta*



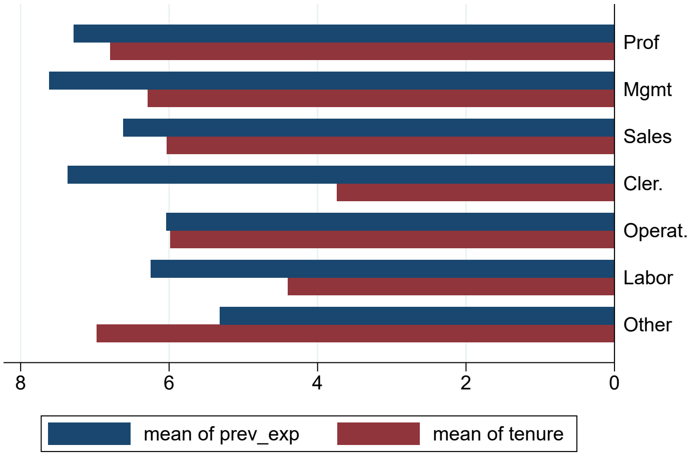
```
graph bar prev_exp tenure, over(occ7) yalternate
```

The `yalternate` option places the y axis on the opposite side, in this example, on the right side of the graph. *Uses nlsw.dta*



```
graph hbar prev_exp tenure, over(occ7) xalternate yreverse
```

We can reverse the direction of the y axis with the `yreverse` option. We combine this with the `xalternate` option to place the labels for the bars on the alternate (right) side of the graph. *Uses nlsw.dta*

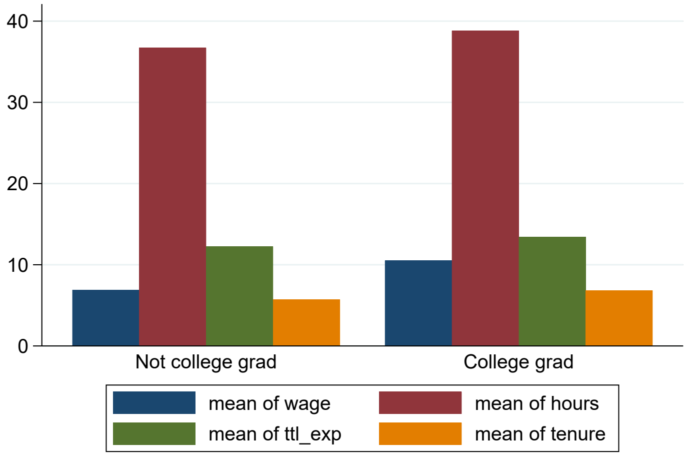

Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other
8
6
4
2
0
mean of prev_exp
mean of tenure

4.8 Changing the look of bars

This section shows how to control the color of the bars and the characteristics of the line outlining the bars. For more information, see the *lookofbar_options* table in `help graph bar` and `help barlook_options`.

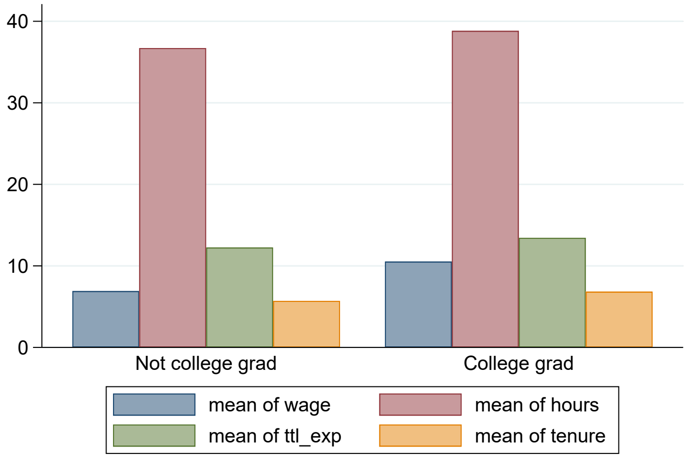
```
graph bar wage hours ttl_exp tenure, over(collgrad)
```

Consider this bar chart. It shows the mean wages, hours worked per week, total experience, and job tenure broken down by whether one graduated college. *Uses nlsw.dta*



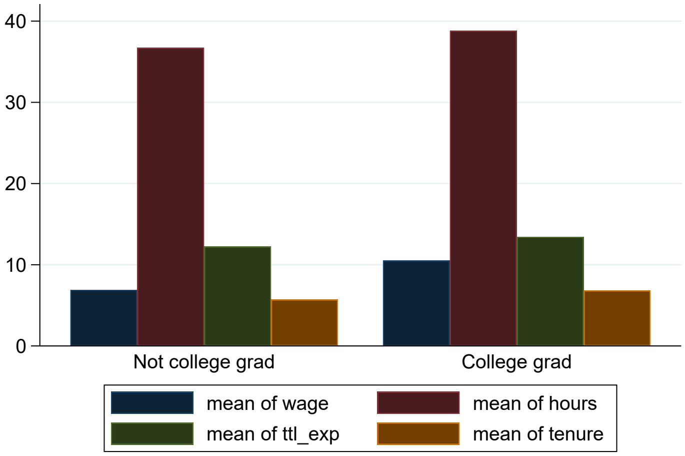
```
graph bar wage hours ttl_exp tenure, over(collgrad)
intensity(*.5)
```

We use the `intensity()` option to control the intensity of the color within the bars. In this example, we request that the color be 50% as intense as it normally would be. *Uses nlsw.dta*



```
graph bar wage hours ttl_exp tenure, over(collgrad)
intensity(*2)
```

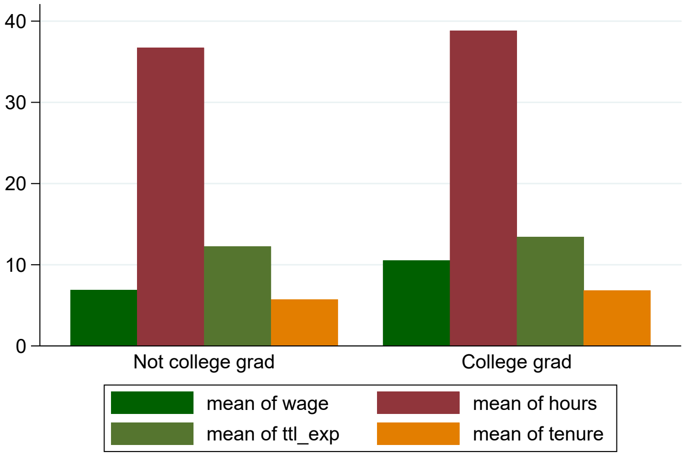
Here we use the `intensity()` option to make the colors within the bars 2 times more intense than they would normally be. Stata also has an option called `lintensity()` that works the same way but controls the intensity of the line surrounding the bar. (This option is not illustrated.) *Uses nlsw.dta*



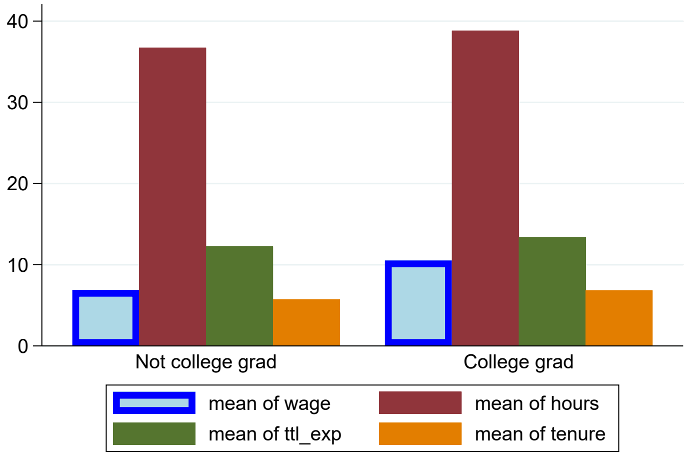
So far, all the options that we have examined determine the overall behavior and look of all the bars as a group. The `bar()` option controls the look of the bars for each y variable, as illustrated below.

```
graph bar wage hours ttl_exp tenure, over(collgrad) bar(1,
bcolor(dkgreen))
```

Here we use the `bar()` option to make the color of the first bar dark green. See **Styles : Colors** (section 10.2) for more information about available colors. *Uses nlsw.dta*

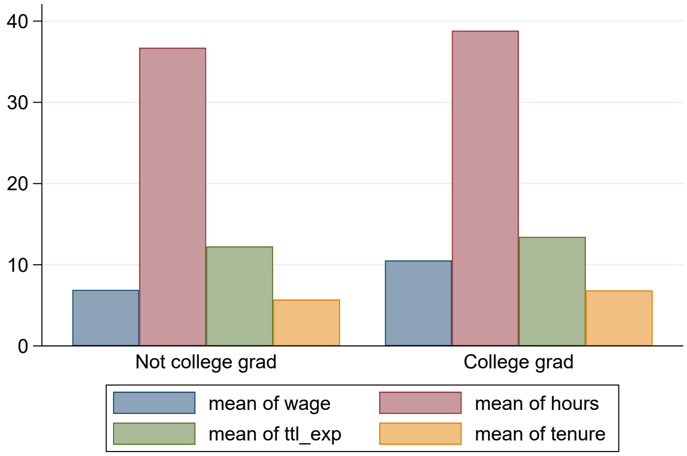


```
graph bar wage hours ttl_exp tenure, over(collgrad) bar(1,
fcolor(ltblue) lcolor(blue) lwidth(vthick))
```



In this example, we make the fill color of the first bar light blue and the outline very thick and blue. See Styles : Linewidth (section 10.7) for more details on controlling the thickness of lines. We could also use the `lpattern()` option to control the pattern of the line surrounding the bar; see Styles : Linepatterns (section 10.6) for more details. *Uses nlsw.dta*

```
graph bar wage hours ttl_exp tenure, over(collgrad)
scheme(vg_palec)
```



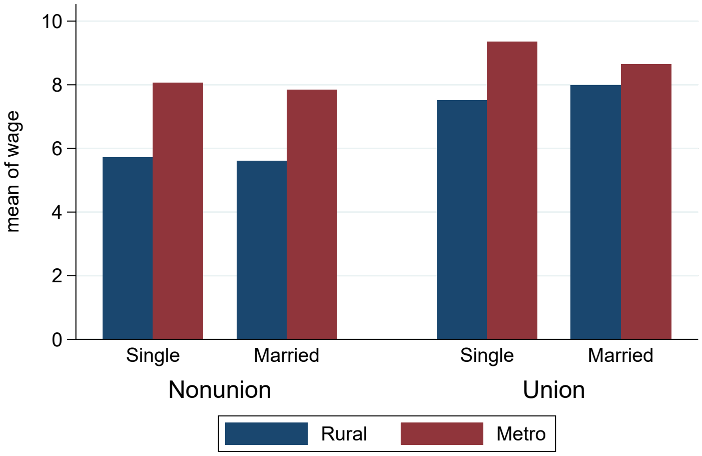
Although we can use the `bar()` option to control the look of each bar, selecting a different scheme allows us to control the look of all the bars. For example, this graph uses the `scheme(vg_palec)` option. See Intro : Schemes (section 1.4) for other schemes and Standard options : Schemes (section 9.2) for more details on schemes. *Uses nlsw.dta*

4.9 Graphing by groups

This section discusses the use of the `by()` option in combination with `graph bar`. Normally, you would use the `over()` option instead of the `by()` option, but there are cases where the `by()` option is either necessary or more advantageous. For example, a `by()` option is useful if you exceed the maximum number of `over()` options (three if you have a single y variable or two if you have multiple y variables). Thus the `by()` option allows you to break down your data by more categorical variables. Also, `by()` gives you more flexibility in the placement of the separate panels. For more information about the `by()` option, see Options : By (section 8.8); for more information about the `over()` option, see Bar : Over (section 4.2).

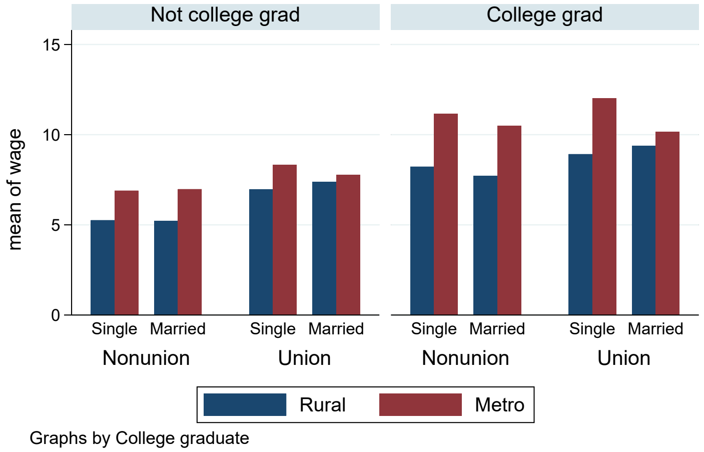
```
graph bar wage, over(urban2) over(married) over(union)
```

Consider this bar graph that breaks down wages by three categorical variables. If we wanted to break down this by another categorical variable, we could not use another `over()` option because we can have a maximum of three `over()` options with one y variable. *Uses nlsw.dta*



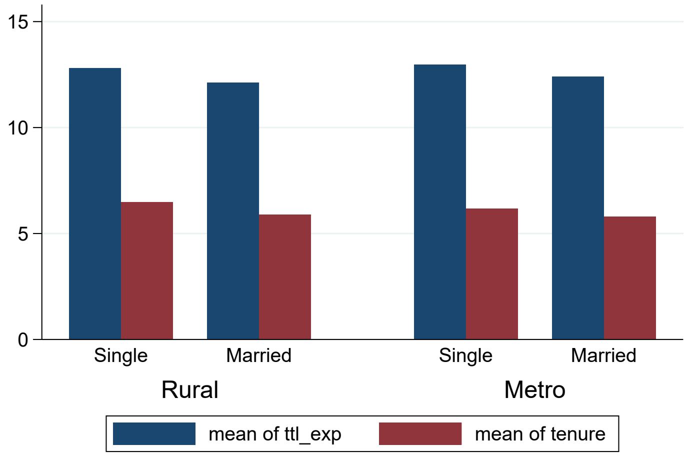
```
graph bar wage, over(urban2) over(married) over(union)
by(collgrad)
```

If we want to show the previous graph separately by `collgrad`, we can use the `by()` option. This option gives us two graphs side by side: one for those who are not college graduates and one for college graduates. *Uses nlsw.dta*



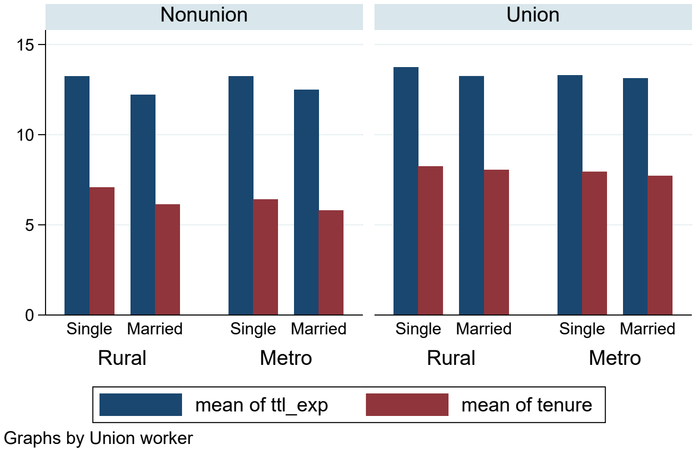
```
graph bar ttl_exp tenure, over(married) over(urban2)
```

Consider this bar graph with multiple *y* variables broken down by two categorical variables by using two `over()` options. When we have multiple *y* variables, we can have a maximum of two `over()` options. *Uses nlsw.dta*



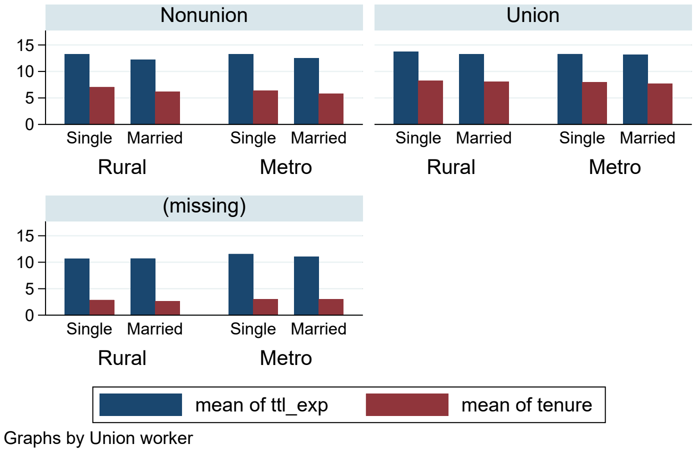
```
graph bar ttl_exp tenure, over(married) over(urban2)
by(union)
```

If we want to show the previous graph by another categorical variable, say, `union`, we can use the `by()` option. *Uses nlsw.dta*



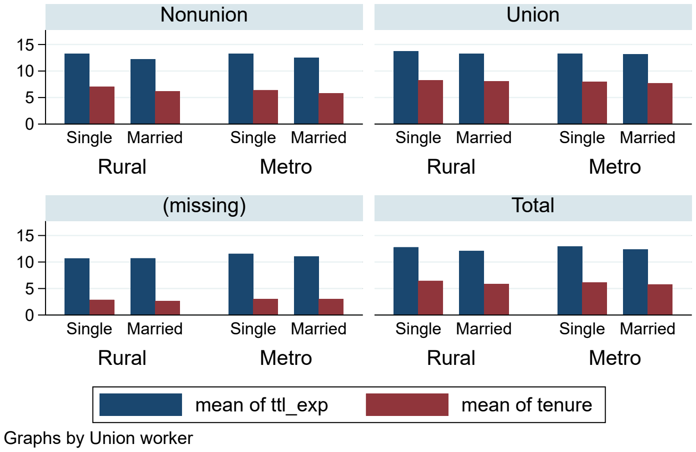
```
graph bar ttl_exp tenure, over(married) over(urban2)
by(union, missing)
```

We can add the `missing` option to include a panel for the missing values of `union`. *Uses nlsw.dta*



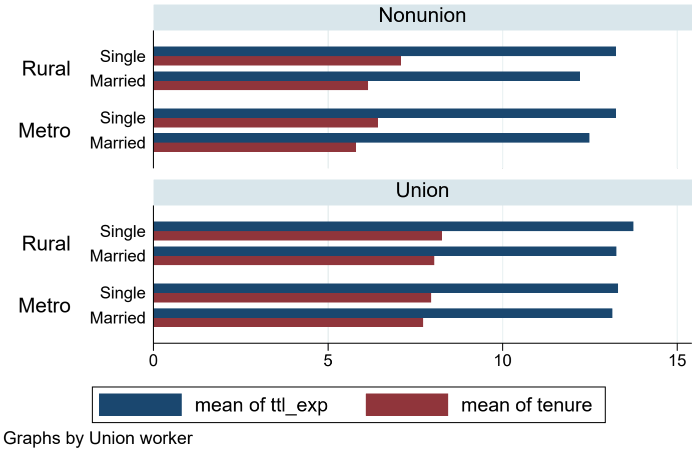
```
graph bar ttl_exp tenure, over(married) over(urban2)
by(union, missing total)
```

We can add the `total` option to include a panel for all observations. *Uses nlsw.dta*



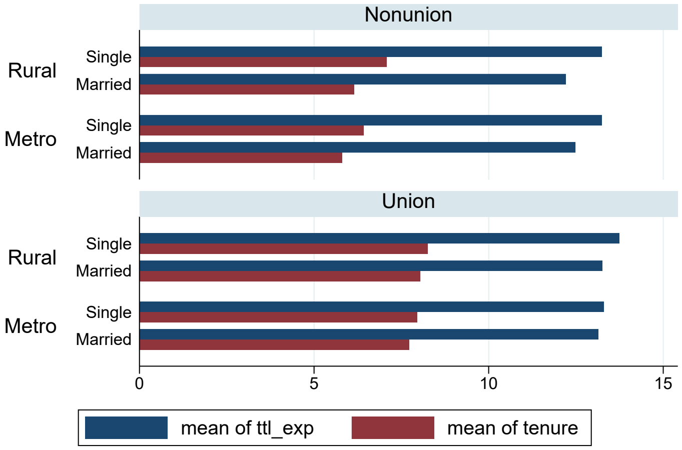
```
graph hbar ttl_exp tenure, over(married) over(urban2)
by(union, cols(1))
```

We remove the `total` and `missing` options and flip the graph to make a horizontal bar chart. We then use the `cols(1)` option to show these graphs in one column. This makes the graph pretty cramped. Let's explore several options we can add to this graph to make it less cramped, adding the options a few at a time. *Uses nlsw.dta*

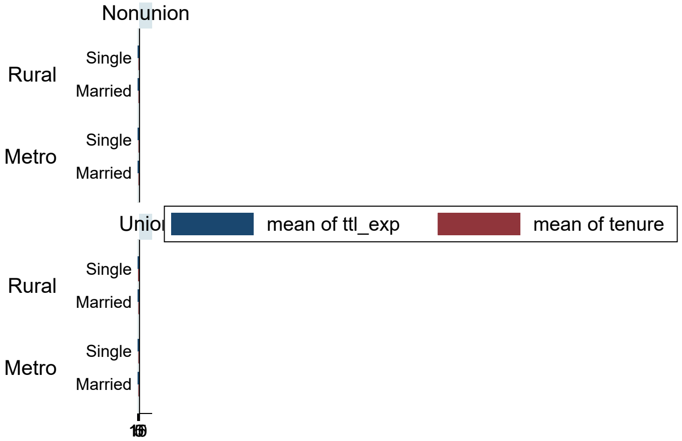


```
graph hbar ttl_exp tenure, over(married) over(urban2)
by(union, cols(1) note(""))
```

We add the `note("")` option within the `by()` option to suppress the note in the left corner, leaving more room for the graph. *Uses nlsw.dta*



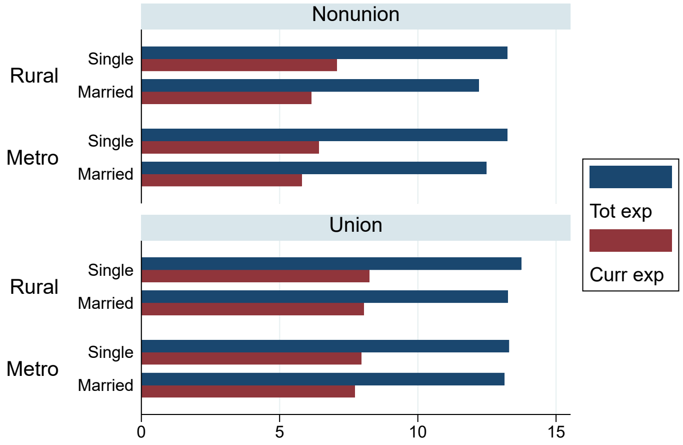
```
graph hbar ttl_exp tenure, over(married) over(urban2)
by(union, cols(1) note("") legend(position(3)))
```



We add the `legend(position(3))` option to put the legend at the right. `legend(position(3))` is contained within the `by()` option because it changes the position of the legend. If we could make the legend narrow (instead of wide), it would work well in this position. *Uses nlsw.dta*

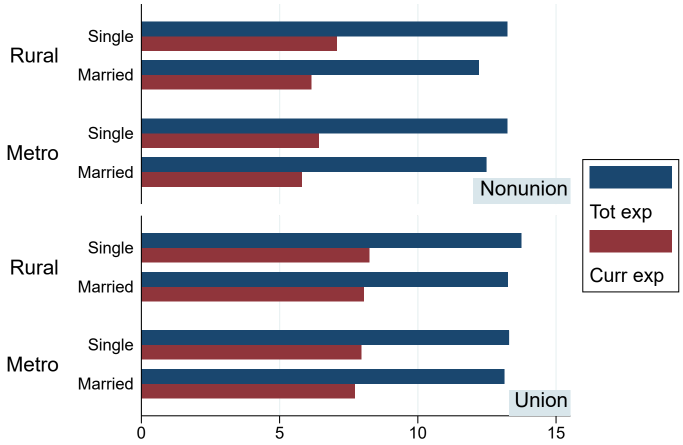
```
graph hbar ttl_exp tenure, over(married) over(urban2)
by(union, cols(1) note("") legend(position(3)))
legend(cols(1) stack label(1 "Tot exp") label(2 "Curr exp"))
```

We add the `legend(cols(1) stack)` option to make the legend narrow and the `label()` option to change the labels in the legend. Here the `legend()` option appears outside the `by()` option. See **Options : By** (section 8.8) and **Options : Legend** (section 8.9) for more information about the interactions of `by()` and `legend()`. *Uses nlsw.dta*



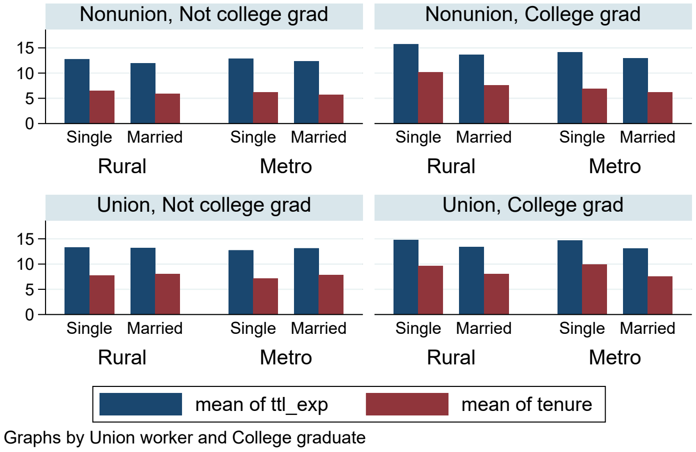
```
graph hbar ttl_exp tenure, over(married) over(urban2)
by(union, cols(1) note("") legend(position(3)))
legend(cols(1) stack label(1 "Tot exp") label(2 "Curr exp"))
subtitle(, position(5) ring(0) nobexpand)
```

We can add the `subtitle()` option to position the title for each graph in the lower right corner. The `position(5)` option puts the title in the five o'clock position, and the `ring(0)` option puts the title inside the plot area. The `nobexpand` (no box expand) option keeps the title from expanding to fill the entire plot area. *Uses nlsw.dta*



```
graph bar ttl_exp tenure, over(married) over(urban2) by(union
collgrad)
```

We can include multiple variables within the `by()` option. Here, in addition to breaking down these variables by two `over()` variables, we break them down by two more variables using the `by(union collgrad)` option. *Uses nlsw.dta*


Nonunion, Not college grad
Nonunion, College grad
Union, Not college grad
Union, College grad
15
10
5
0
Single
Married
Rural
Metro
mean of ttl_exp
mean of tenure
Graphs by Union worker and College graduate

Chapter 5
Box plots

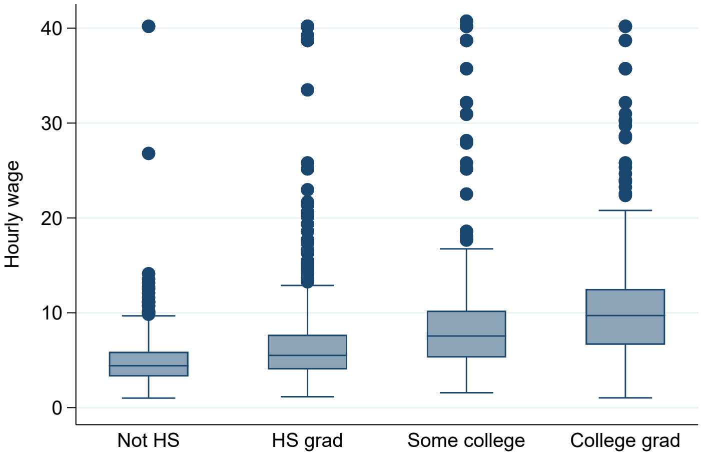
A box plot displays a box(es) bordered at the 25th and 75th percentiles of the y variable with a *median line* at the 50th percentile. Whiskers extend from the box to the upper and lower *adjacent values* and are capped with an *adjacent line*. Values exceeding the upper and lower adjacent values are called *outside values* and are displayed as markers.

This chapter explores how to create box plots using the `graph box` command. The first section illustrates how you can create box plots as a function of categorical variables (specified by the `over()` option) and how you can create box plots from multiple y variables. The next section illustrates how you can control the gaps among boxes along the categorical axis. Then I illustrate options for sorting the boxes, including options for sorting them alphabetically and sorting them on the median. Next I show how you can customize the labeling of the categorical axis, including customizing the content of the labels, the size and color of the labels, and the gaps between the labels. The next section illustrates how you can customize the display of the box plot graph legend, including how to modify the labels and position the legend. Next I illustrate options allowing you to control the y axis, including options for customizing the y-axis title and the y-axis labels and even for hiding the y axis altogether. The next section illustrates ways to customize the display of the boxes and whiskers. The chapter concludes by showing how to use the `by()` option to create multiple graphs as a function of the levels of the *by-variable* specified in the `by()` option. All examples in this chapter will use the `nlsw` data file.

5.1 Specifying variables and groups

This section introduces the use of box plots, illustrating the use of the `over()` option for showing box plots by one or more grouping variables. We will look at examples showing how we can graph multiple variables at once by specifying more y variables, followed by some examples of general options for controlling the display of multiple y variables and the behavior of the `over()` options. See the *group_options* table in `help graph box` for more details.

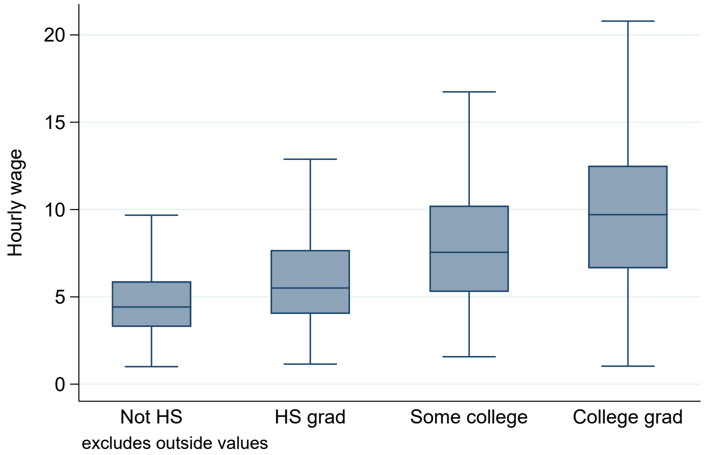
```
graph box wage, over(grade4)
```



This box plot uses the `over(grade4)` option to break down wages by education level (in four categories). By default, the separate levels of `grade4` are graphed using the same color, and the levels are labeled on the x axis. The graph shows many outside values that are displayed as markers beyond the whiskers. The following example shows how we can suppress the display of the outside values. *Uses nlsw.dta*

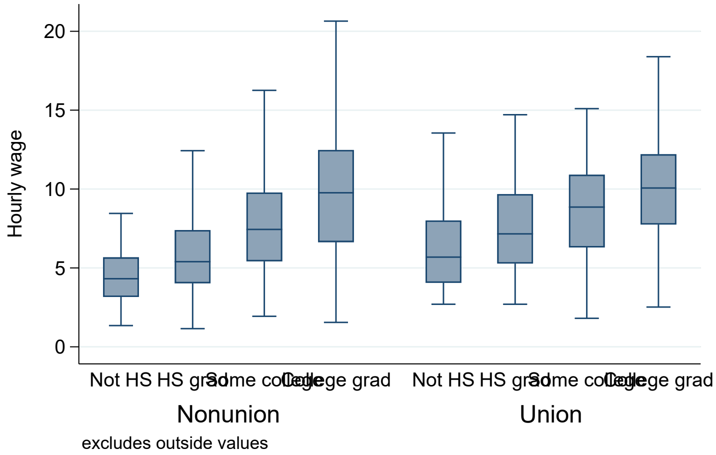
```
graph box wage, over(grade4) nooutsides
```

By adding the `nooutsides` option, we suppress the display of the outside values. Graphs using this option have a note in the bottom left corner indicating that the outside values have been excluded from display in the graph. For most of the graphs in this chapter, there would be many outside values, which would make the graphs cluttered, so most of the graphs will use the `nooutsides` option. *Uses nlsw.dta*



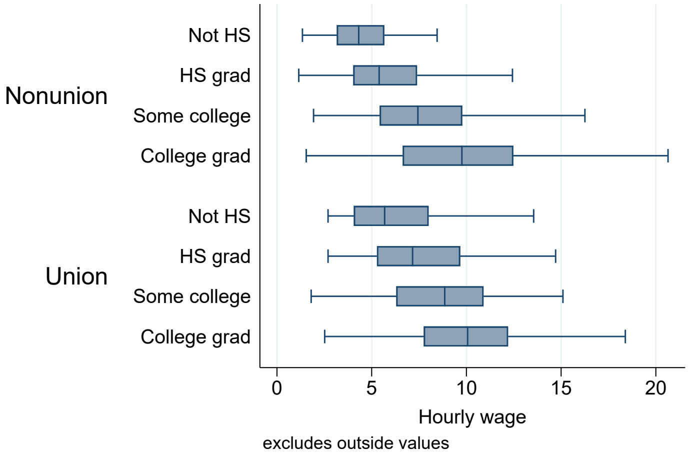
```
graph box wage, nooutsides over(grade4) over(union)
```

In this example, we add the `over(union)` option to show wages broken down by education and whether one is a member of a union. However, the labels for `grade4` overlap each other. See the next example for one solution. *Uses nlsw.dta*



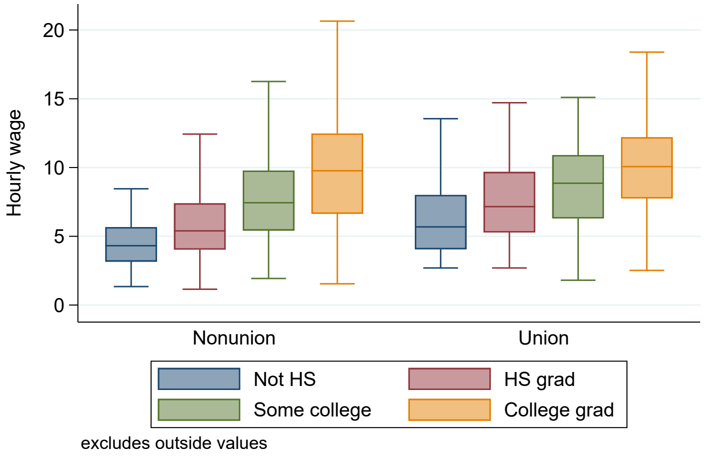
```
graph hbox wage, nooutsides over(grade4) over(union)
```

Here we use `graph hbox` to make a horizontal box plot. This eliminates the overlapping of the labels for `grade4`. The next example will show another possible solution. *Uses nlsw.dta*



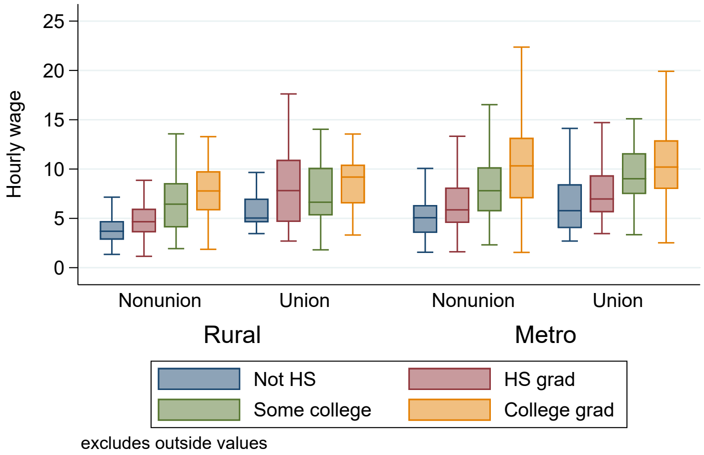
```
graph box wage, nooutsides over(grade4) over(union) asyvars
```

With the `asyvars` option, the first `over()` variable, `grade4`, is treated as though it were multiple y variables. As a result, the levels of `grade4` are shown in multiple colors and labeled with a legend. We can only use `asyvars` when we have one y variable. *Uses nlsw.dta*



```
graph box wage, nooutsides over(grade4) over(union)
over(urban2)
```

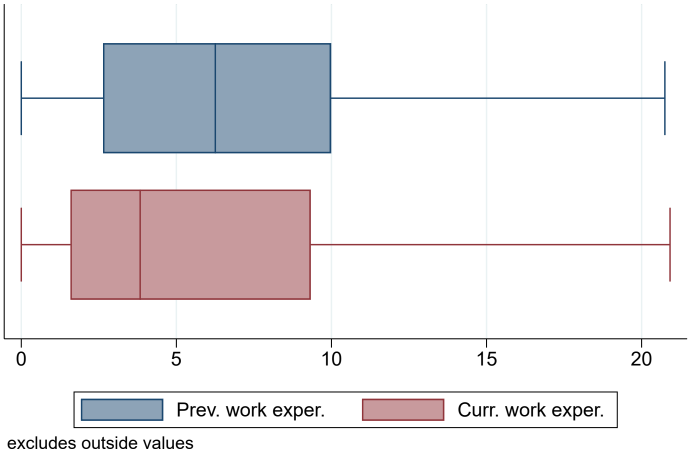
Here we add a third `over()` option, comparing people who live in rural and metropolitan areas. The first `over()` variable, `grade4`, is now treated as though it were multiple y variables. Because of this, we can specify only one y variable when we have three `over()` options. *Uses nlsw.dta*



Now let's look at examples of using multiple y variables with the `over()` option, starting with a graph with multiple y variables.

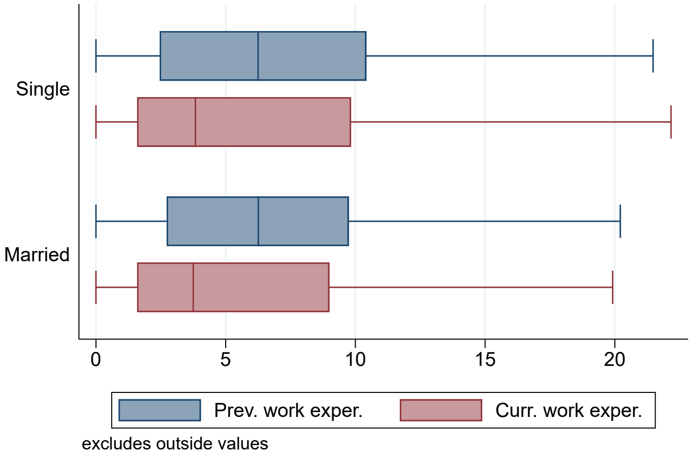
```
graph hbox prev_exp tenure, nooutsides
```

This graph shows work experience before one's current job and work experience at one's current job. *Uses nlsw.dta*



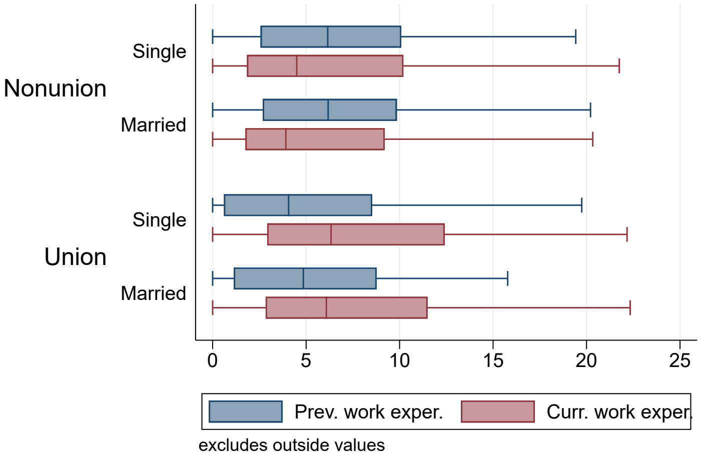
```
graph hbox prev_exp tenure, nooutsides over(married)
```

We can further break down these variables by marital status. *Uses nlsw.dta*



```
graph hbox prev_exp tenure, nooutsides over(married)
over(union)
```

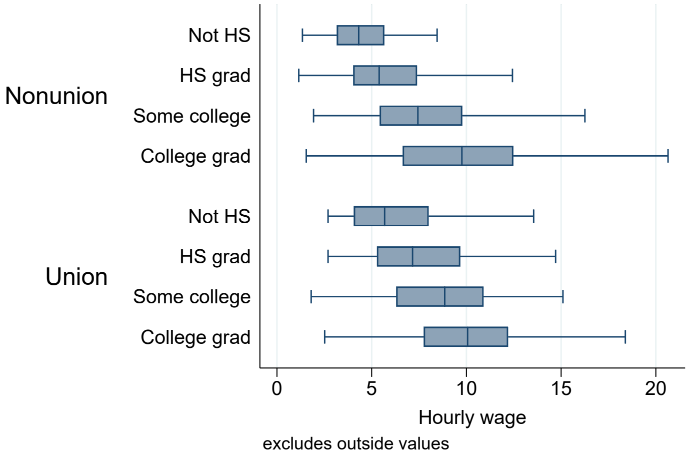
We can take the last graph and add another `over()` option to even further break down these variables by whether one belongs to a union. However, we cannot add a third `over()` option when we have multiple y variables, but we could add the `by()` option; see Box : By (section 5.8). *Uses nlsw.dta*



Now let's consider options that may be used in combination with the `over()` option to customize the behavior of the graphs. We will first explore how you can treat the levels of the first `over()` option as though they were multiple y variables. We will then see how you can request that missing values for the levels of the `over()` variables be displayed and how you can suppress empty categories when multiple `over()` options are used.

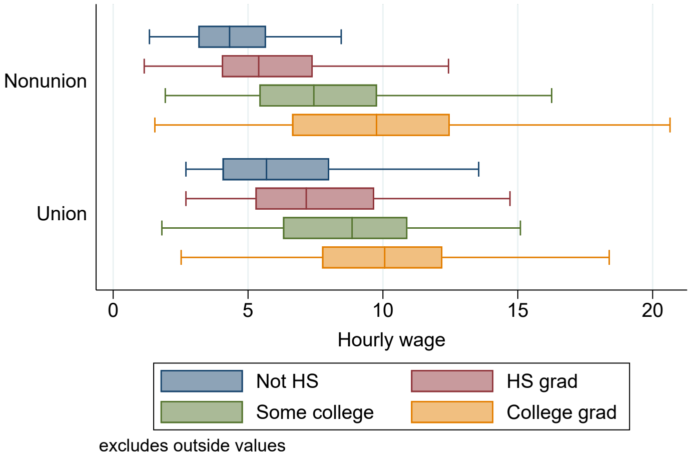
```
graph hbox wage, nooutsides over(grade4) over(union)
```

Consider this graph where we show wages broken down by education level and whether one belongs to a union. *Uses nlsw.dta*



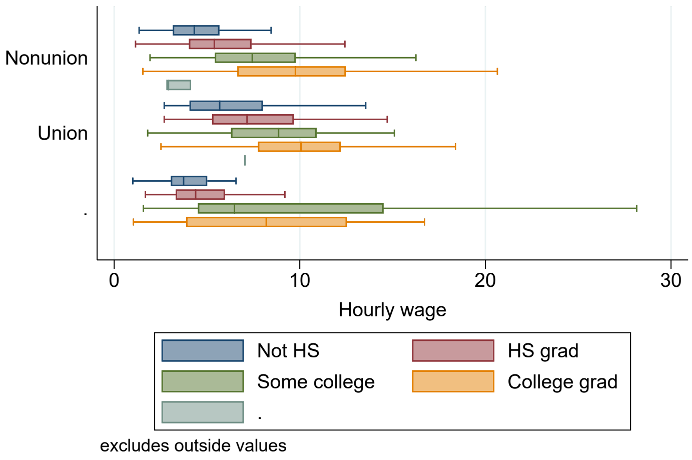
```
graph hbox wage, nooutsides over(grade4) over(union) asyvars
```

If we add the `asyvars` option, the first `over()` variable (`grade4`) is graphed as if there were four y variables corresponding to each level of `grade4`. Each level of `grade4` is shown as a differently colored/shaded box and labeled with the legend. *Uses nlsw.dta*



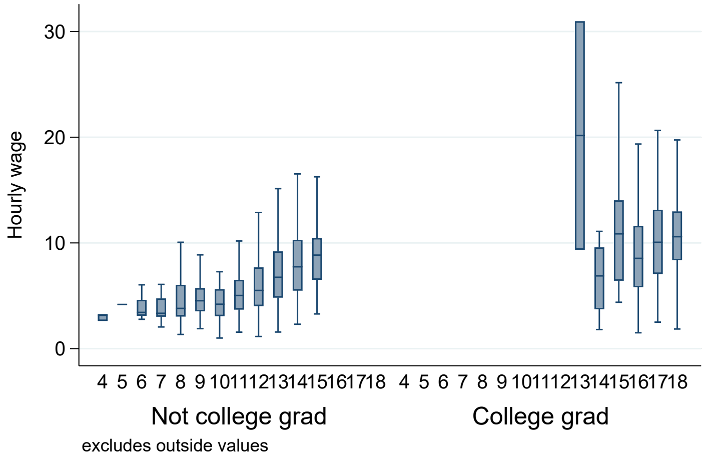
```
graph hbox wage, nooutsides over(grade4) over(union) asyvars
missing
```

By adding the `missing` option to the previous graph, we see a category for those who are missing on the `union` variable, shown as the third group, which is labeled with a dot to indicate that those values are missing; see Box : Cat axis (section 5.4) to see how we could label this differently (for example, labeling it with the word “Missing”). *Uses nlsw.dta*



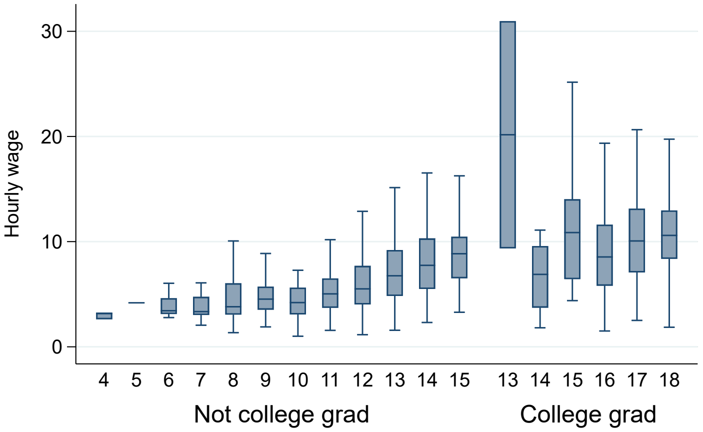
```
graph box wage, nooutsides over(grade) over(collgrad)
```

Consider this box chart that breaks down wages by two variables: the last grade that one completed and whether one is a college graduate. By default, Stata shows all possible combinations for these two variables. Usually, all combinations are possible, but not here. *Uses nlsw.dta*



```
graph box wage, nooutsides over(grade) over(collgrad) nofill
```

If we want to display only the combinations of the `over()` variables that exist in the data, we can use the `nofill` option. *Uses nlsw.dta*



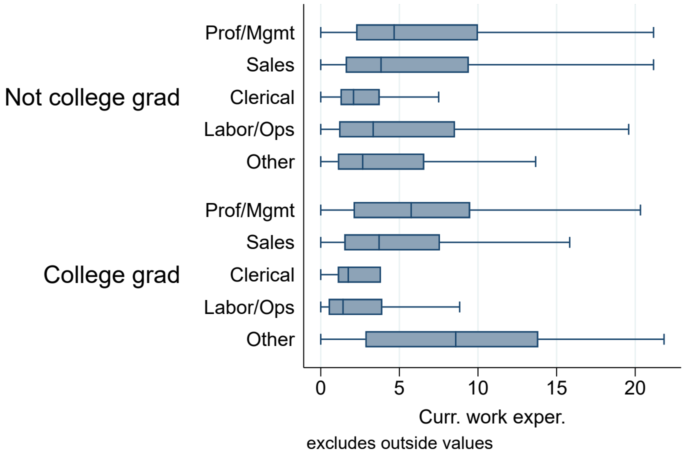
excludes outside values

5.2 Options for controlling gaps between boxes

This section considers some of the options that we can use to control the gaps (spacing) between boxes. The options for controlling gaps between boxes work exactly the same as the options to control gaps between bars with the `graph bar` command. These options were illustrated in more detail in Bar : Gaps (section 4.3), so I refer you to that section for further exploration.

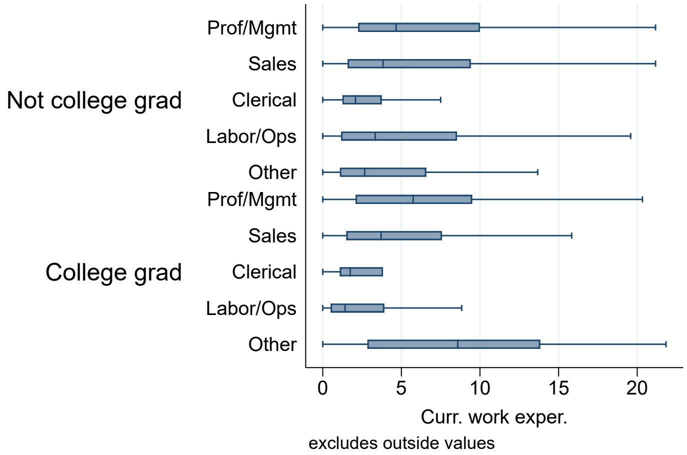
```
graph hbox tenure, nooutsides over(occ5) over(collgrad)
```

Consider this graph that shows box plots of `tenure` broken down by `occ5` and `collgrad`. *Uses nlsw.dta*



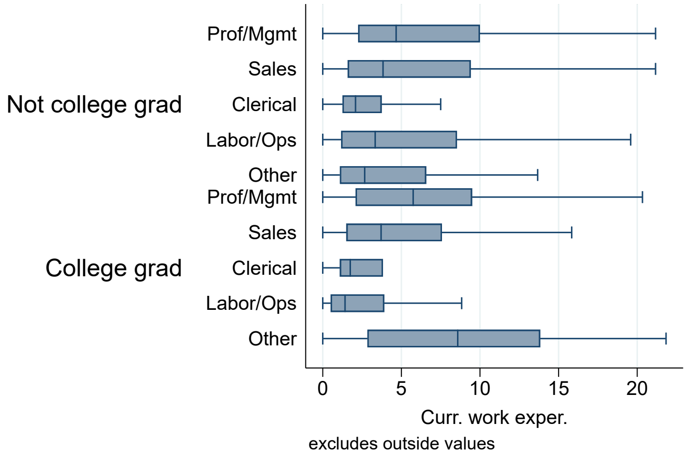
```
graph hbox tenure, nooutsides over(occ5, gap(300))
over(collgrad)
```

By default, the gap among levels `occ5` would be two-thirds (67%) the width of a box. With the `gap(300)` option, that gap is increased to three times the width of a box. *Uses nlsw.dta*



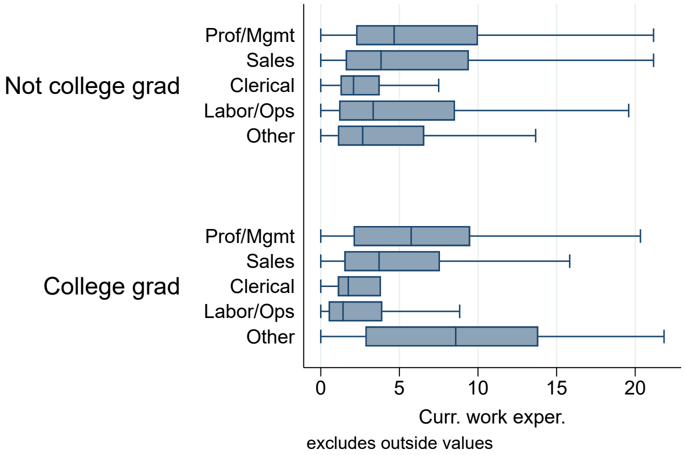
```
graph hbox tenure, nooutsides over(occ5) over(collgrad,
gap(25))
```

Here we shrink the gap between the levels of `collgrad`, making the gaps 25% of the width of a box, yielding boxes that are wider than they normally would be. *Uses nlsw.dta*



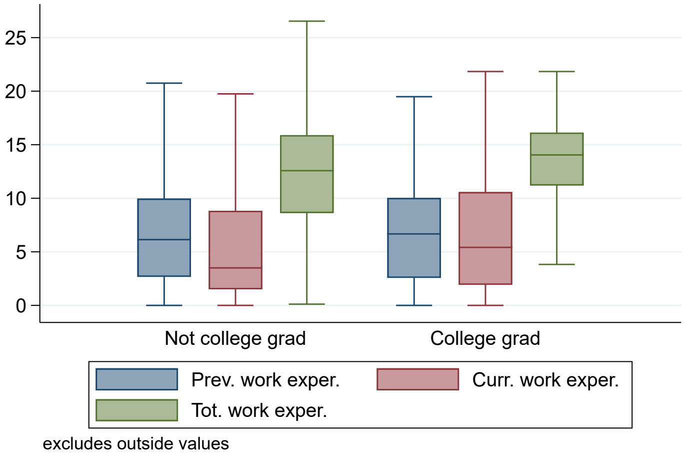
```
graph hbox tenure, nooutsides over(occ5, gap(25))
over(collgrad, gap(400))
```

We can control the gap with respect to each `over()` variable. In this example, we make the gap among the `occ5` categories small (25% of the width of a box) and the gap between the levels of `collgrad` large (four times the width of a box). *Uses nlsw.dta*



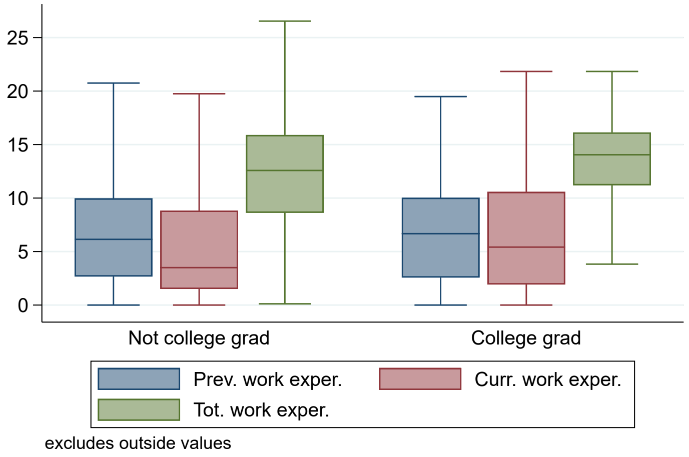
```
graph box prev_exp tenure ttl_exp, nooutsides over(collgrad)
outergap(150)
```

We can change the outer gap between the boxes and the edge of the plot area with the `outergap()` option. Here the gap is 1.5 times the size of a box. *Uses nlsw.dta*



```
graph box prev_exp tenure ttl_exp, nooutsides over(collgrad)
boxgap(10)
```

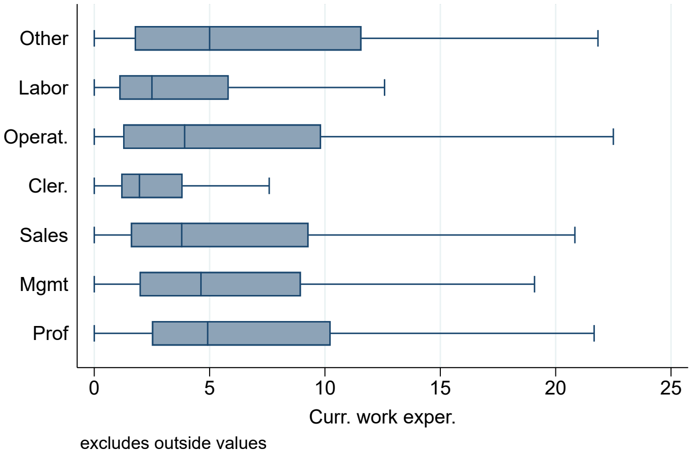
The `boxgap()` option controls the size of the gap among the boxes formed by the multiple y variables. The default value is 33, meaning that the distance between the boxes is 33% of the width of the boxes. Here we make the gap only 10% of the width of a box. *Uses nlsw.dta*


25
20
15
10
5
0
Not college grad
College grad
Prev. work exper.
Curr. work exper.
Tot. work exper.
excludes outside values

5.3 Options for sorting boxes

This section considers some of the options that we can use to control the order in which the boxes are displayed. By default, the boxes formed by `over()` variables are ordered in ascending sequence according to the values of the `over()` variable. Stata allows us to control the order of the boxes by allowing us to put them in descending order, order them according to the values of another variable, or sort the boxes according to their medians.

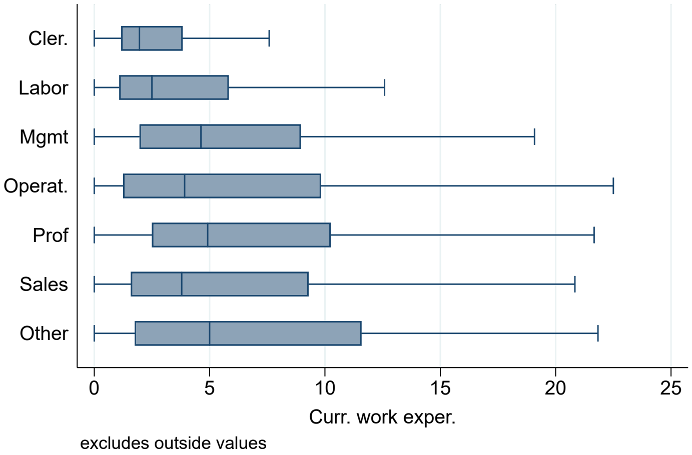
```
graph hbox tenure, nooutsides over(occ7, descending)
```



Consider this graph showing `tenure` broken down by the seven levels of occupation. The boxes would normally be ordered by levels of `occ7`, going from 1 to 7. The `descending` option switches the order of the boxes. They still are ordered according to the seven levels of occupation, but the boxes are ordered going from 7 to 1. *Uses nlsw.dta*

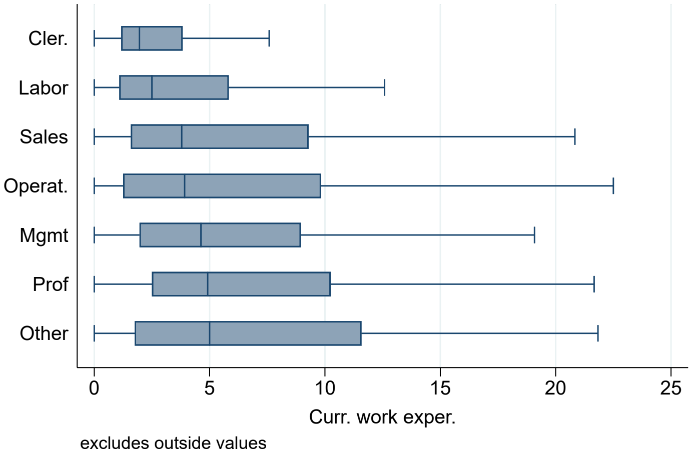
```
graph hbox tenure, nooutsides over(occ7, sort(occ7alpha))
```

We might want to put these boxes in alphabetical order, but with *Other* still appearing last. We can do this by recoding `occ7` into a new variable (say, `occ7alpha`) such that, as `occ7alpha` goes from 1 to 7, the occupations are alphabetically ordered. We recoded `occ7` with these assignments: $4 = 1$, $6 = 2$, $2 = 3$, $5 = 4$, $1 = 5$, $3 = 6$, and $7 = 7$. Then the `sort(occ7alpha)` option alphabetizes the boxes but with *Other* still appearing last. *Uses nlsw.dta*



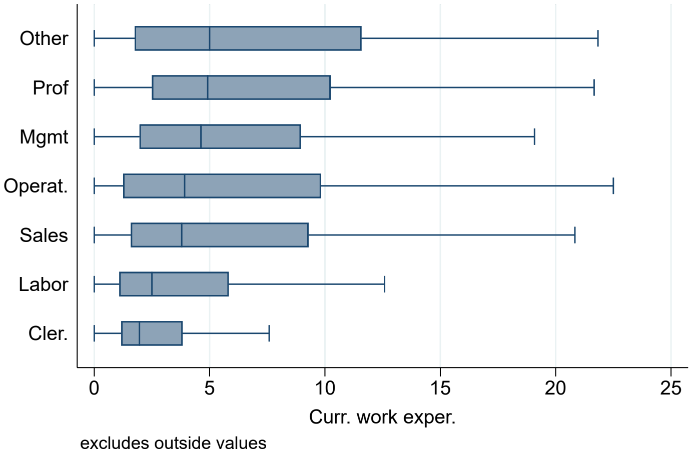
```
graph hbox tenure, nooutsides over(occ7, sort(1))
```

Here we sort the variables on the median of `tenure`, yielding boxes with medians in ascending order. The `sort(1)` option sorts the boxes according to the median of the first y variable, meaning to sort on the median of `tenure`. *Uses nlsw.dta*



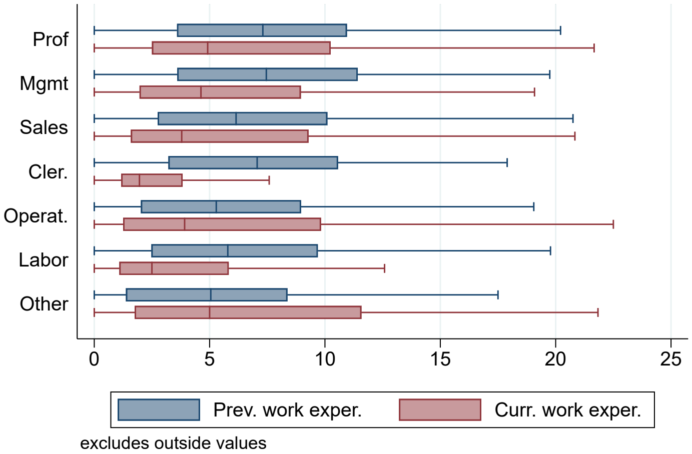
```
graph hbox tenure, nooutsides over(occ7, sort(1) descending)
```

Adding the `descending` option yields boxes in descending order, going from highest median `tenure` to lowest median `tenure`. *Uses nlsw.dta*



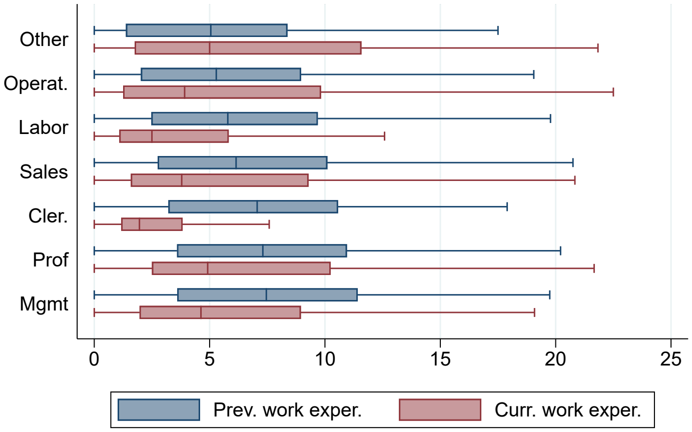
```
graph hbox prev_exp tenure, nooutsides over(occ7)
```

Here we plot two y variables: the number of years of work experience before one's current job and the years in one's current job. Because we have removed all `sort()` options, the boxes are sorted according to the values of `occ7`. *Uses nlsw.dta*



```
graph hbox prev_exp tenure, nooutsides over(occ7, sort(1))
```

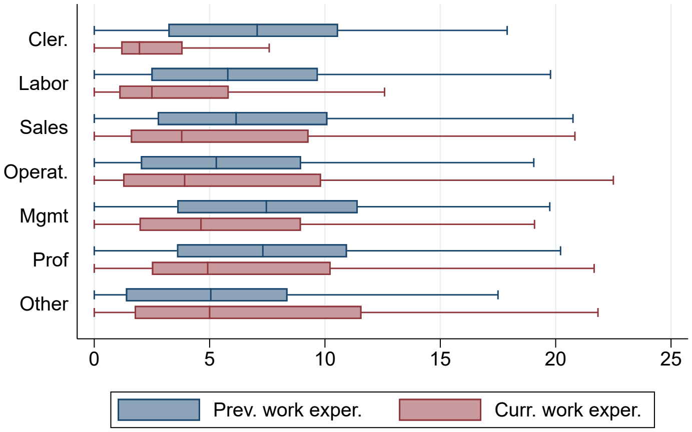
Adding the `sort(1)` option now sorts the boxes according to the median of `prev_exp` because that is the first y variable. *Uses nlsw.dta*



excludes outside values

```
graph hbox prev_exp tenure, nooutsides over(occ7, sort(2))
```

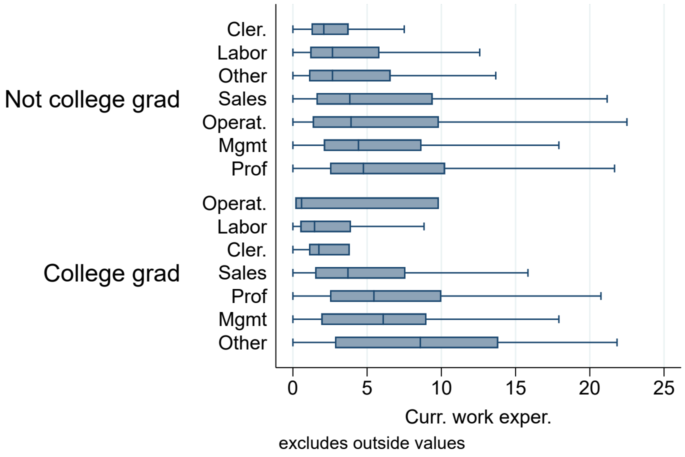
Changing `sort(1)` to `sort(2)` now sorts the boxes according to the median of the second y variable, `tenure`. *Uses nlsw.dta*



excludes outside values

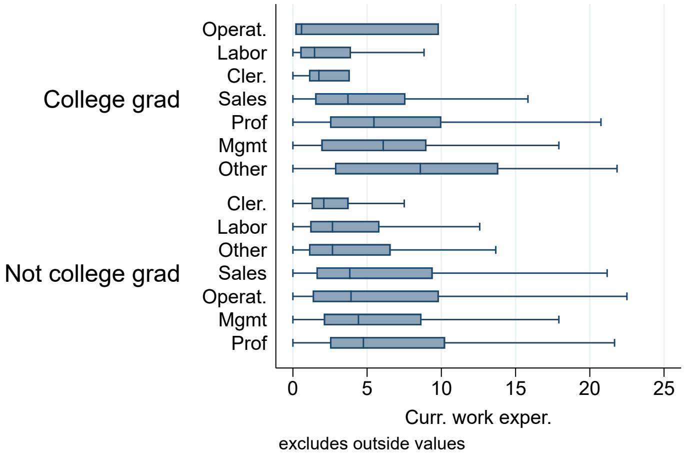
```
graph hbox tenure, nooutsides over(occ7, sort(1))
over(collgrad)
```

We can use the `sort()` option when there are more `over()` variables. Here the boxes are ordered according to the median of `tenure` across `occ7` but within each level of `collgrad`. *Uses nlsw.dta*



```
graph hbox tenure, nooutsides over(occ7, sort(1))
over(collgrad, descending)
```

We add the `descending` option to the second `over()` option, and the levels of `collgrad` are now shown with college graduates appearing first. *Uses nlsw.dta*



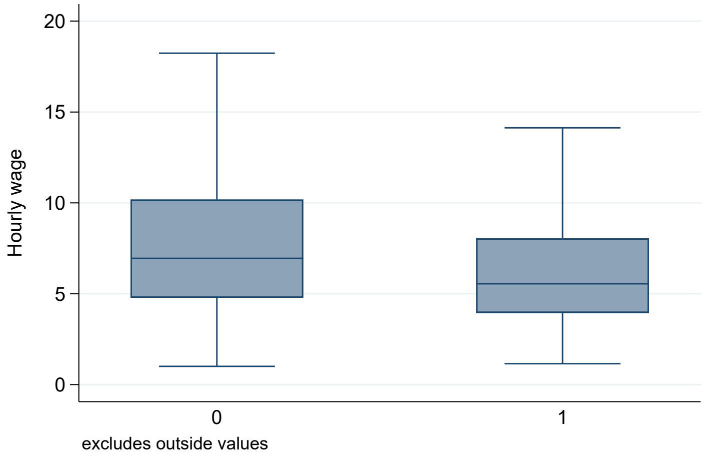
5.4 Controlling the categorical axis

This section describes ways that we can label categorical axes. Box plots are similar to bar charts, but they are different from other graphs because their x axes are represented by categorical variables. This section describes options we can use to customize these categorical axes. For more details, see `help cat_axis_label_options` and `help cat_axis_line_options`.

This section begins by showing examples of how we can change the labels for the x axis for these categorical variables.

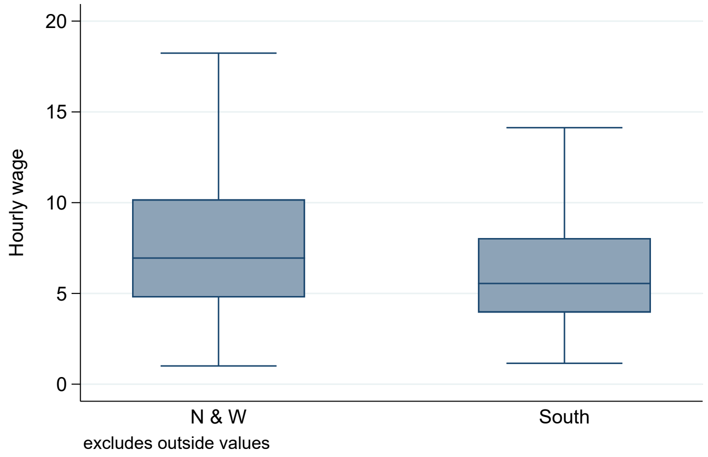
```
graph box wage, nooutsides over(south)
```

This is an example of a box plot with one `over()` variable graphing wages broken down by whether one lives in the South. The variable `south` is a dummy variable that does not have any value labels, so the x axis is not labeled well. *Uses nlsw.dta*



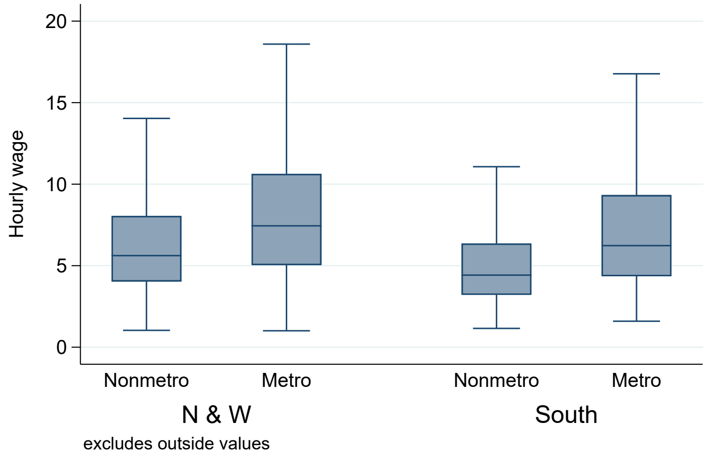
```
graph box wage, nooutsides over(south, relabel(1 "N & W" 2
"South"))
```

The `relabel()` option changes the labels displayed for the levels of `south`, giving the x axis more meaningful labels. Note that we wrote `relabel(1 "N & W")`, not `relabel(0 "N & W")`, because these numbers do not represent the actual levels of `south` but the ordinal position of the levels, that is, first and second. *Uses nlsw.dta*



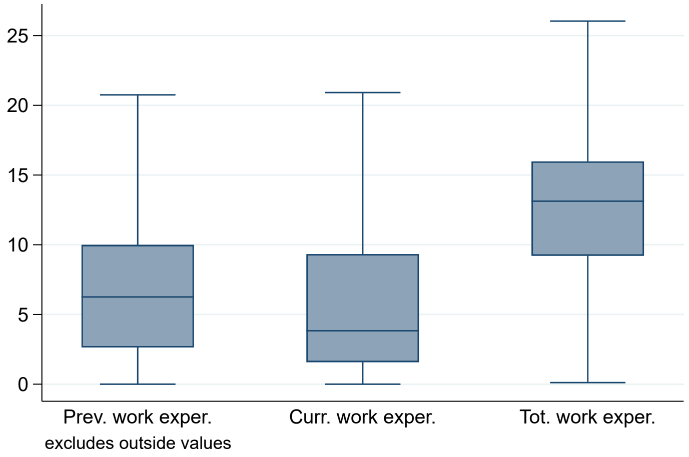
```
graph box wage, nooutsides over(smsa, relabel(1 "Nonmetro" 2
"Metro")) over(south, relabel(1 "N & W" 2 "South"))
```

This is an example of a box plot with two `over()` variables. Here we use the `relabel()` option to change the labels displayed for the levels of `south` and `smsa`. *Uses nlsw.dta*



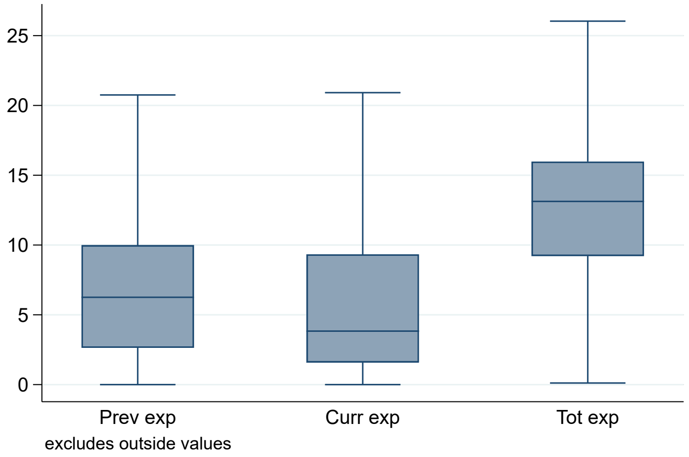
```
graph box prev_exp tenure ttl_exp, nooutsides ascategory
```

This box plot shows multiple y variables, but we use the `ascategory` option to plot the different y variables as if they were categorical variables. The boxes for the different variables are the same color, and the categories are labeled on the x axis rather than with a legend. The default labels on the x axis are not bad, but we might want to change them. *Uses nlsw.dta*



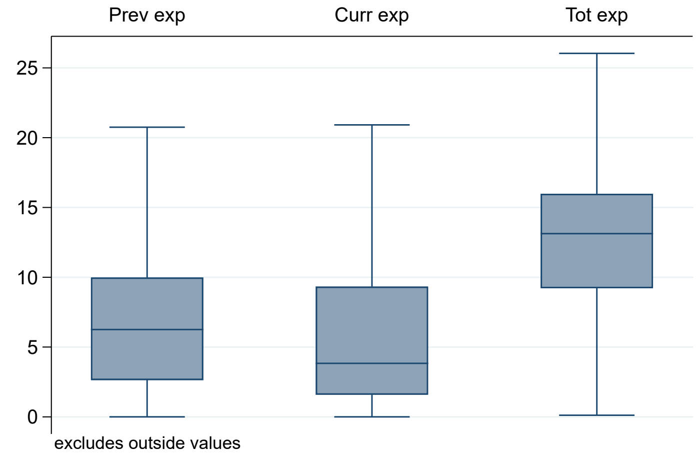
```
graph box prev_exp tenure ttl_exp, nooutsides ascategory
yvaroptions(relabel(1 "Prev exp" 2 "Curr exp" 3 "Tot exp"))
```

If we had an `over()` option, we would use the `relabel()` option to change the labels on the x axis. But because we had multiple y variables that we have treated as categories, we use the `yvaroptions(relabel())` option to modify the labels on the x axis. *Uses nlsw.dta*



```
graph box prev_exp tenure ttl_exp, nooutsides ascategory
yvaroptions(relabel(1 "Prev exp" 2 "Curr exp" 3 "Tot exp"))
xalternate
```

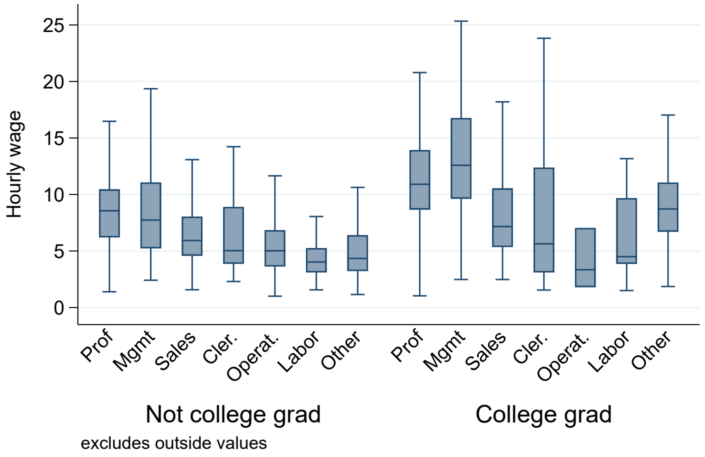
We move the x axis to the opposite side of the graph using the `xalternate` option. We can also use the `yalternate` option to move the y axis to its opposite side. *Uses nlsw.dta*



We have seen that although the `relabel()` option is called an `over()` option, we can use it within `yvaroptions()` to control the labeling of multiple y variables (provided that the `ascategory` option is used to convert the multiple y variables into categories). We next explore other `over()` options that we can use with `yvaroptions()`.

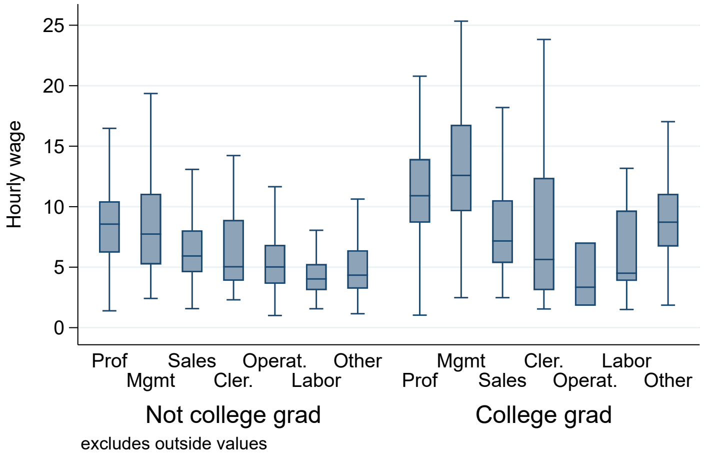
```
graph box wage, nooutsides over(occ7, label(angle(45)))
over(collgrad)
```

This graph shows wages broken down by occupation and by whether one graduated college. We add the `label(angle(45))` option to rotate the labels for occupation by 45 degrees. If this had been omitted, the labels would have overlapped each other. *Uses nlsw.dta*



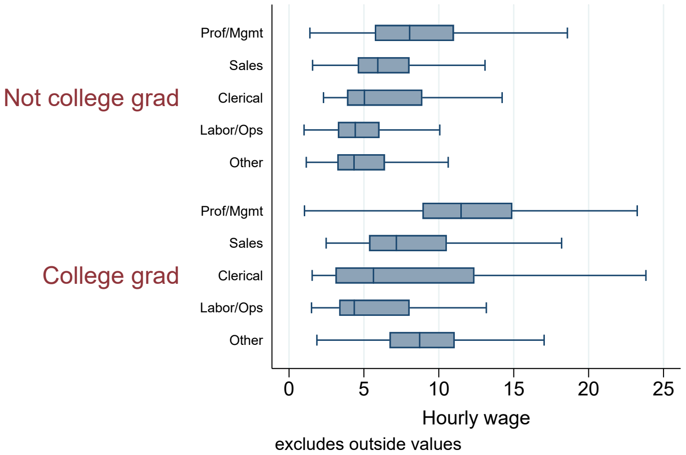
```
graph box wage, nooutsides over(occ7, label(alternate))
over(collgrad)
```

Compare this graph with the previous example. This example uses the `label(alternate)` strategy to avoid overlapping by alternating the labels for occupation. *Uses nlsw.dta*



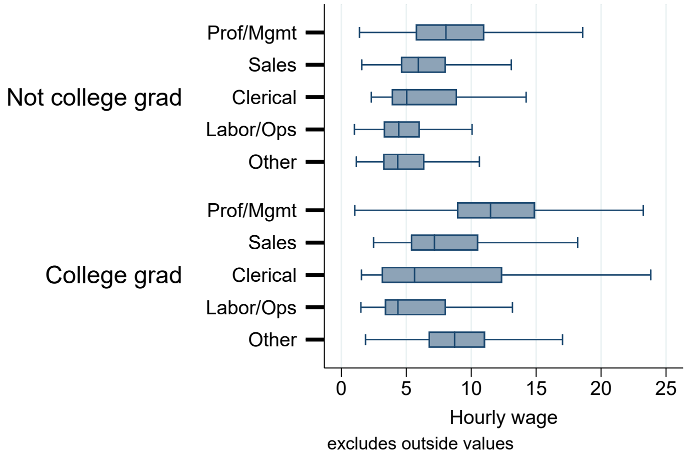
```
graph hbox wage, nooutsides over(occ5, label(labsize(small)))
over(collgrad, label(labcolor(maroon)))
```

Here we make the labels for `occ5` small and the color of the labels for `collgrad` maroon. See Styles : Colors (section 10.2) for more color options and Styles : Textsize (section 10.12) for more size options (including sizing using points, inches, and centimeters). *Uses nlsw.dta*

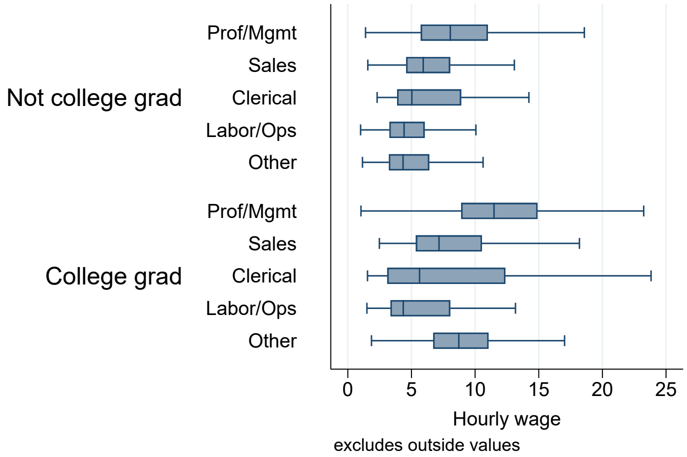


```
graph hbox wage, nooutsides over(occ5, label(ticks
tlwidth(thick) tlength(medium))) over(collgrad)
```

The `ticks` option adds tick marks. Here we make them thick in width and medium in length. See `help cat_axis_label_options` for more details. *Uses nlsw.dta*



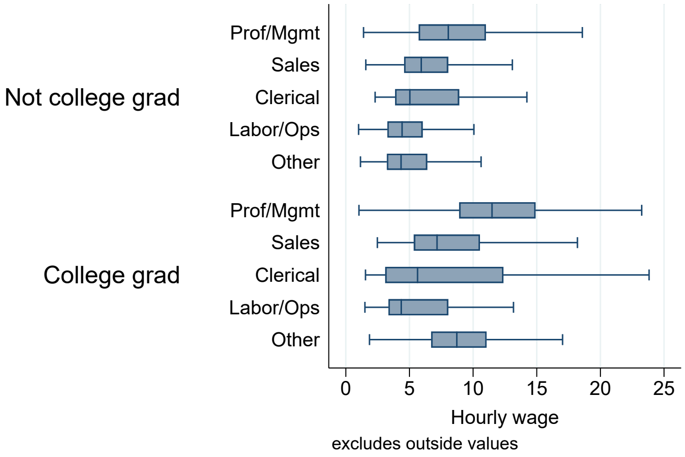
```
graph hbox wage, nooutsides over(occ5, label(labgap(huge)))
over(collgrad)
```



The `label(labgap(huge))` option controls the gap between the label and the axis line. Here we increase the gap between the label for the levels of `occ5` and the axis line to huge. *Uses nlsw.dta*

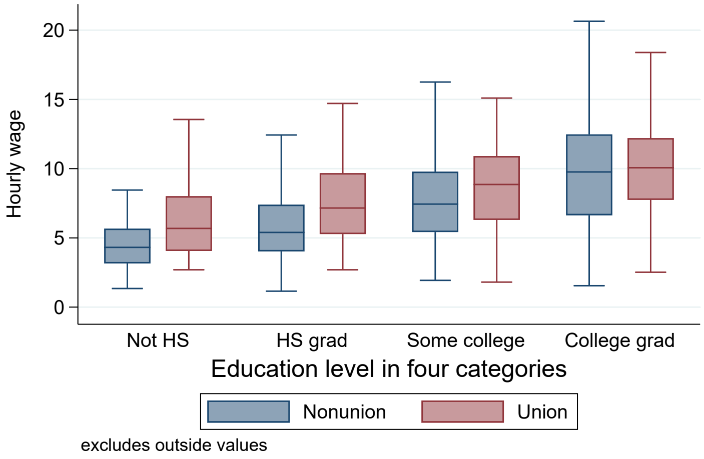
```
graph hbox wage, nooutsides over(occ5) over(collgrad,
label(labgap(huge)))
```

By using the `label(labgap(huge))` option, we increase the gap associated with `collgrad`. This example makes the gap between `collgrad` and `occ5` huge. *Uses nlsw.dta*



```
graph box wage, nooutsides over(union) over(grade4) asyvars
b1title("Education level in four categories")
```

The `b1title()` option adds a title to the bottom of the graph, in effect labeling the x axis. We can add a second title below that by using the `b2title()` option. If we used `graph hbox`, we could label the left axis using the `l1title()` and `l2title()` options. *Uses nlsw.dta*

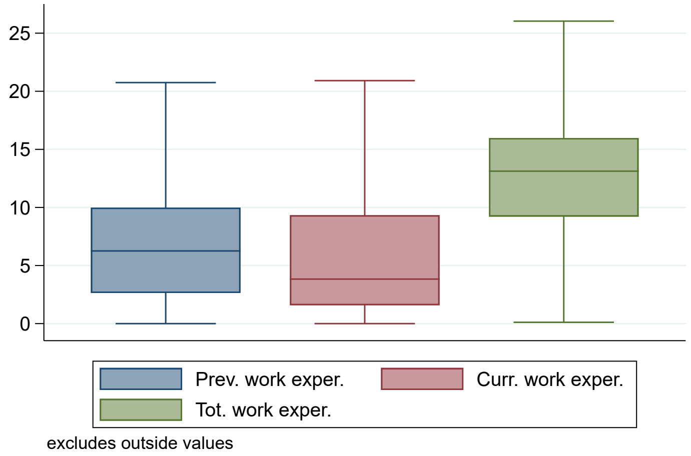


5.5 Controlling the legend

This section discusses the use of the legend for box charts, emphasizing the features that are unique to box charts. The section Options : Legend (section 8.9) goes into great detail about the legend, as does `help legend_options`. We can use the legend for multiple y variables or when the first `over()` variable is treated as a y variable with the `asyvars` option. See Box : Yvars and over (section 5.1) for more information about using multiple y variables and more examples of treating the first `over()` variable as a y variable.

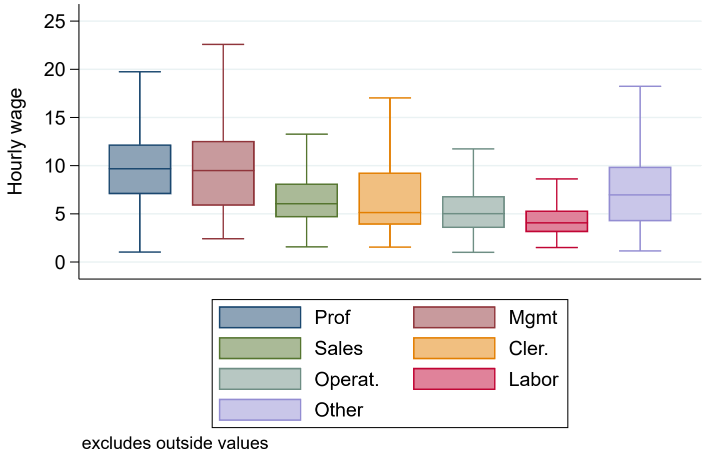
```
graph box prev_exp tenure ttl_exp, nooutsides
```

Consider this box plot of three different variables. These variables are shown with different colors, and a legend is used to identify the variables. *Uses nlsw.dta*



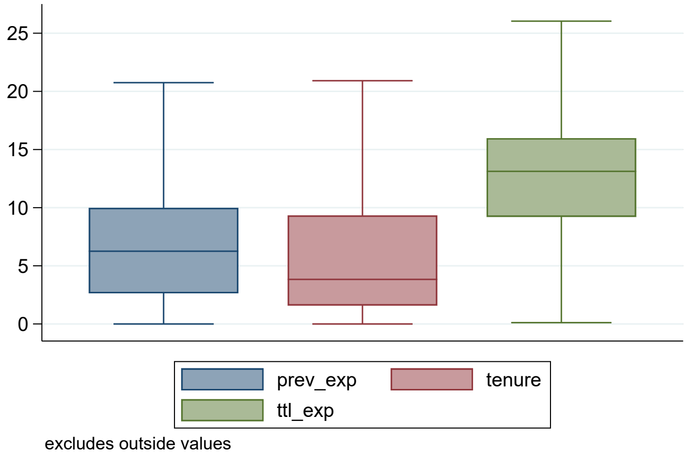
```
graph box wage, nooutsides over(occ7) asyvars
```

This is another example of where a legend can arise in a Stata box plot by using the `asyvars` option, which treats an `over()` variable as though the levels were different y variables. *Uses nlsw.dta*



Unless otherwise mentioned, the `legend()` option described below works the same whether we derived the legend from multiple y variables or from an `over()` option that we combined with the `asyvars` option.

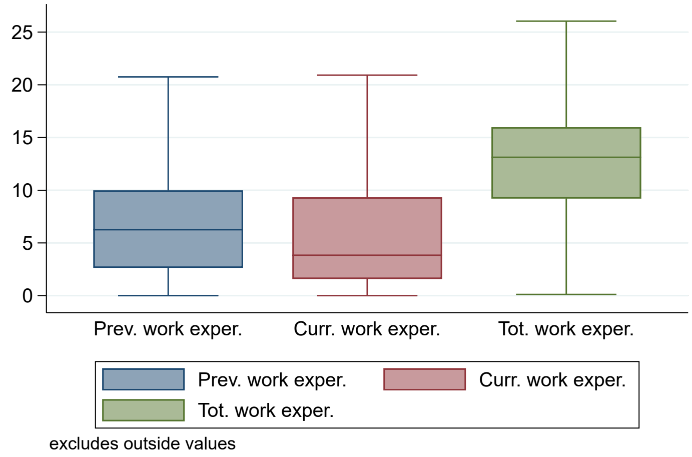
```
graph box prev_exp tenure ttl_exp, nooutsides nolabel
```



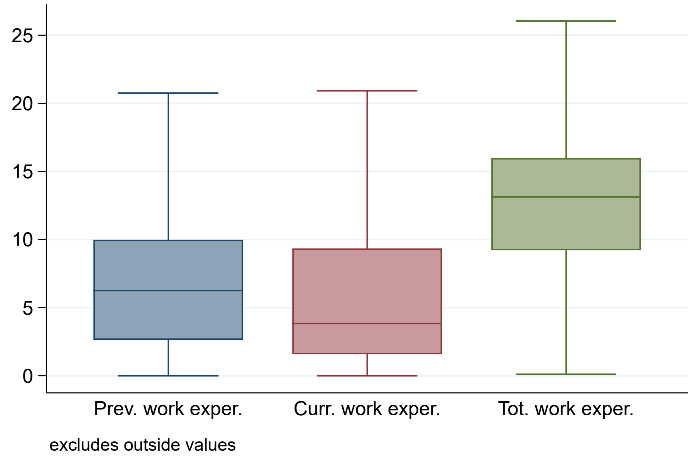
The `nolabel` option works only when we have multiple y variables. When we use this option, the variable names (not the variable labels) are used in the legend. For example, instead of showing the variable label `Prev. work exper.`, it shows the variable name `prev_exp`. *Uses nlsw.dta*

```
graph box prev_exp tenure ttl_exp, nooutsides showyvars
```

The `showyvars` option puts the labels in the legend. *Uses nlsw.dta*



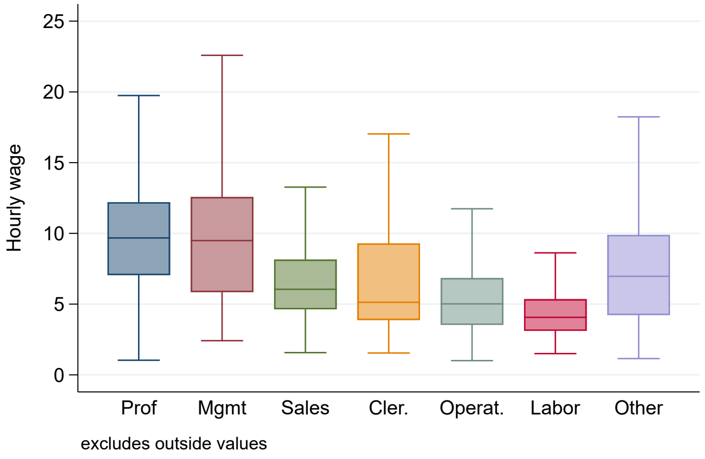
```
graph box prev_exp tenure ttl_exp, nooutsides showyvars
legend(off)
```



One instance when the `showyvars` option would be useful is when we want separately colored boxes labeled at the bottom. Here we use `showyvars` to show the labels at the bottom of the boxes and the `legend(off)` option to suppress the display of the legend. *Uses nlsw.dta*

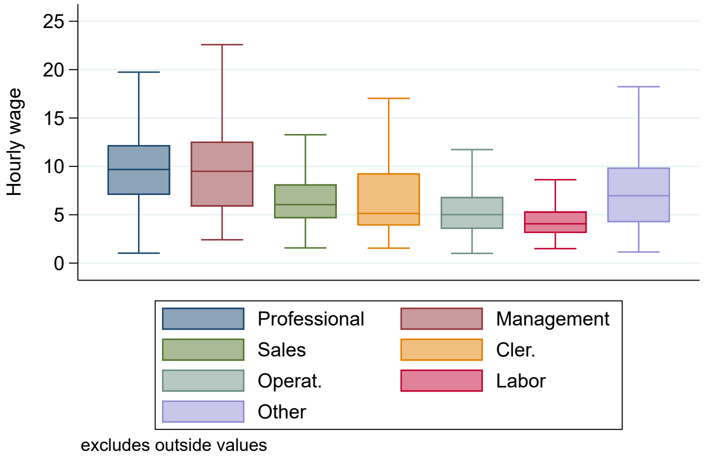
```
graph box wage, nooutsides over(occ7) asyvars showyvars
legend(off)
```

Even though the `showyvars` option sounds like it would work only with multiple y variables, it also works when we combine the `over()` and `asyvars` options. As before, we suppress the legend by using the `legend(off)` option. *Uses nlsw.dta*



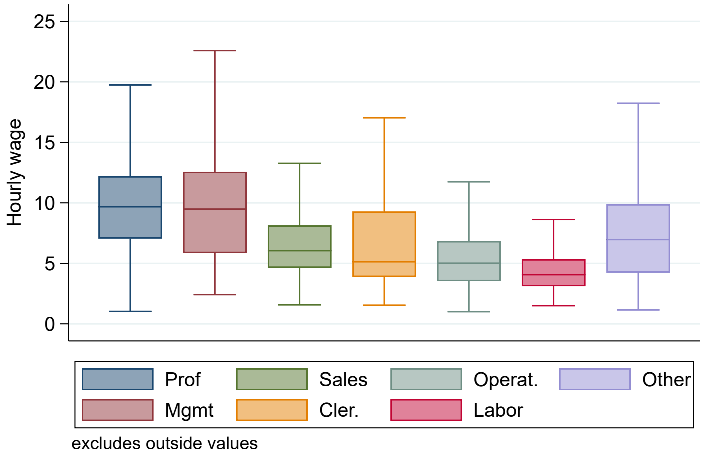
```
graph box wage, nooutsides over(occ7) asyvars legend(label(1
"Professional") label(2 "Management"))
```

We use the `legend(label())` option to change the labels for the first and second variables in the legend. We use a separate `label()` option for each box. *Uses nlsw.dta*



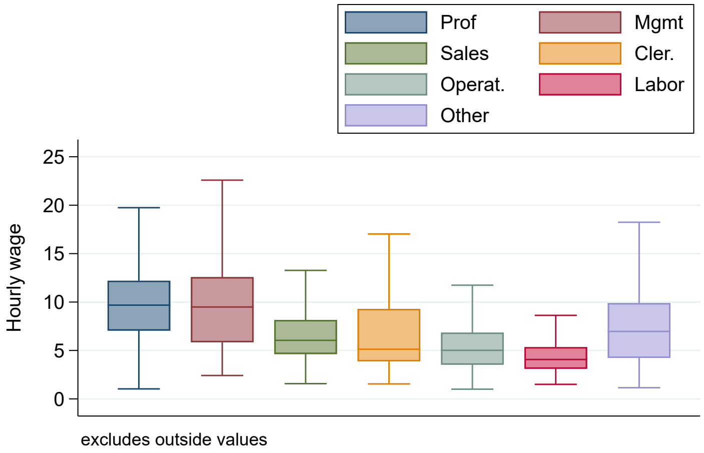
```
graph box wage, nooutsides over(occ7) asyvars legend(rows(2)
colfirst)
```

The `rows(2)` option combined with `colfirst` displays the legend in two rows with the keys ordered by column (instead of by row, the default). This yields keys that are more adjacent to the bars that they label. *Uses nlsw.dta*



As we can see, the default placement for the legend is below the x axis. However, Stata gives us tremendous flexibility in the placement of the legend. Let's now consider options that control the placement of the legend, along with options useful for controlling the placement of the items within the legend.

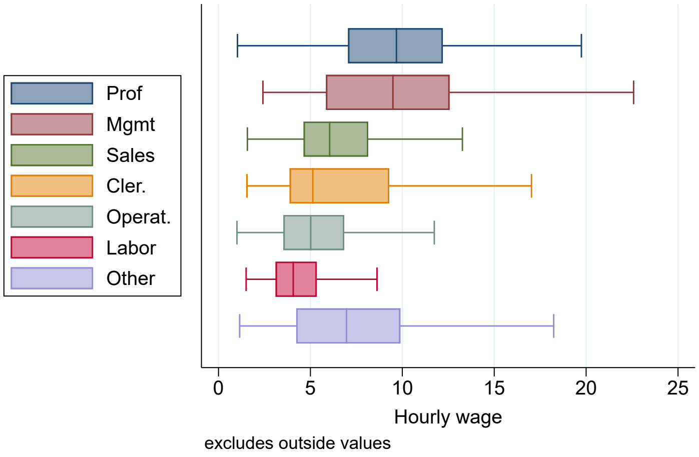
```
graph box wage, nooutsides over(occ7) asyvars
legend(position(1))
```



We can use the `legend(position(1))` option to place the legend in the top right corner of the graph. The values we supply for `position()` are like the numbers on a clock face, where twelve o'clock is the top, six o'clock is the bottom, and 0 represents the center of the clock face. Specifying one o'clock places the legend in the top right; see **Styles : Clockpos** (section 10.3) for more details. *Uses nlsw.dta*

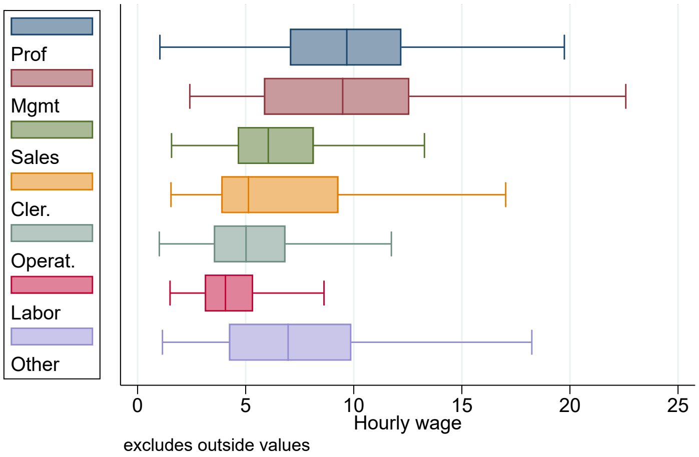
```
graph hbox wage, nooutsides over(occ7) asyvars legend(cols(1)
position(9))
```

We switch to making this a horizontal box plot and move the legend by using the `position(9)` option to place the legend in the nine o'clock position. We also use the `cols(1)` option to display the legend as one column. *Uses nlsw.dta*



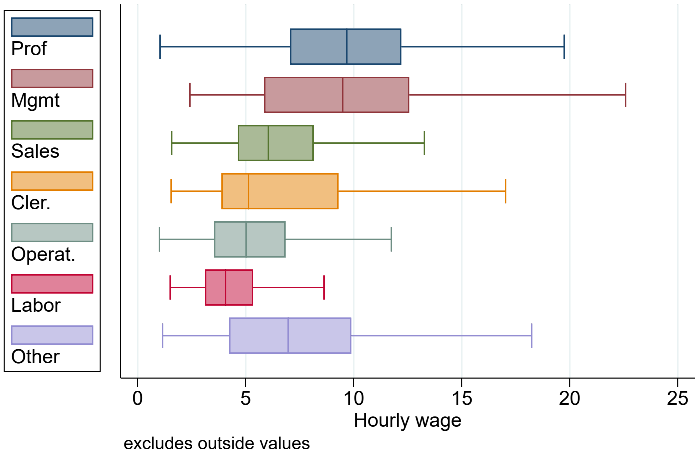
```
graph hbox wage, nooutsides over(occ7) asyvars legend(cols(1)
position(9) stack)
```

The `stack` option stacks the keys and their labels. Other options for controlling the legend include `rowgap()`, `keygap()`, `symxsize()`, `symysize()`, `textwidth()`, and `symplacement()`. See Options : Legend (section 8.9) and `help legend_options` for more details. *Uses nlsw.dta*



```
graph hbox wage, nooutsides over(occ7) asyvars
scheme(vg_lgndc)
```

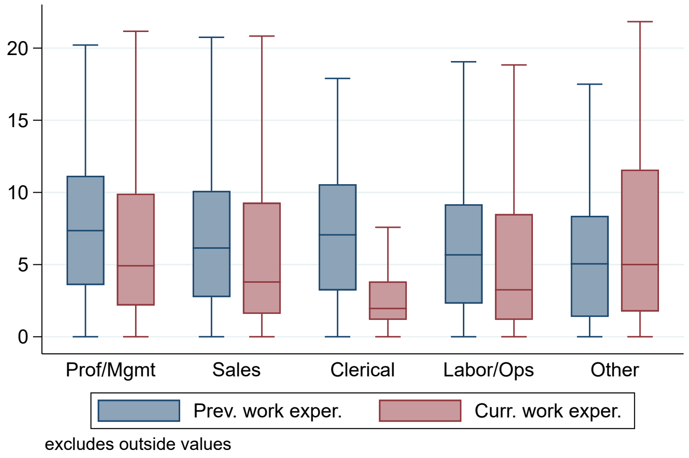
We now specify the `scheme(vg_lgndc)` to select the `vg_lgndc` scheme. This scheme positions the legend at the left in one column. *Uses nlsw.dta*


Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other
0
5
10
15
20
25
Hourly wage
excludes outside values

5.6 Controlling the y axis

This section describes options we can use with respect to the y axis with box charts. To be precise, when Stata refers to the y axis on a box chart, it refers to the axis with the continuous variable, whether that be the left axis when using `graph box` or the bottom axis when using `graph hbox`. This section emphasizes the features that are particularly relevant to box charts. For more details, see Options : Axis titles (section 8.4), Options : Axis labels (section 8.5), and Options : Axis scales (section 8.6). See also `help axis_title_options`, `help axis_label_options`, and `help axis_scale_options`. A number of these examples are shown using the `vg_lgndc` scheme, which places the legend to the left in one column.

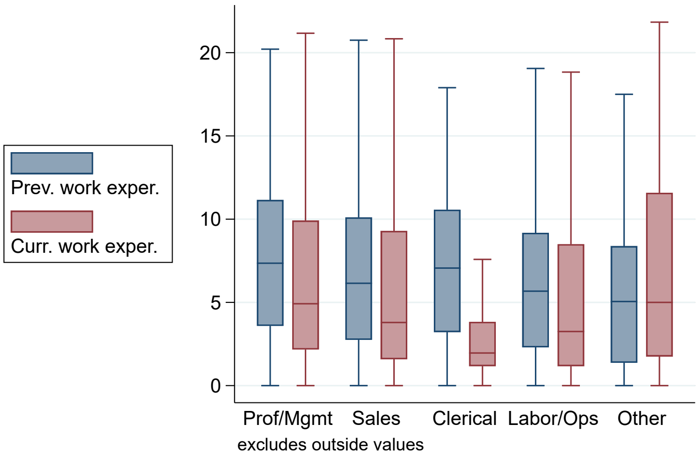
```
graph box prev_exp tenure, nooutsides over(occ5)
```



Consider this graph showing previous and current work experience broken down by occupation. We use the `nooutsides` option to suppress the display of outside values. For the rest of the graphs in this section, there would be many outside values, which would make the graphs cluttered, so we will continue to use the `nooutsides` option for each example. *Uses nlsw.dta*

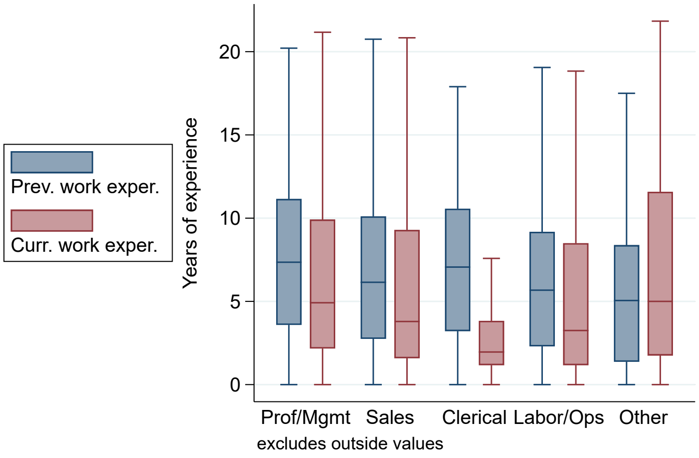
```
graph box prev_exp tenure, nooutsides over(occ5)
scheme(vg_lgndc)
```

Consider the impact of adding the `scheme(vg_lgndc)` option. We could have added a number of options to have achieved the positioning and organization of the legend, but using `scheme(vg_lgndc)` gave us those changes via the addition of that one option. See Standard options : Schemes (section 9.2) for additional details about schemes. *Uses nlsw.dta*



```
graph box prev_exp tenure, nooutsides over(occ5)
ytitle("Years of experience") scheme(vg_lgndc)
```

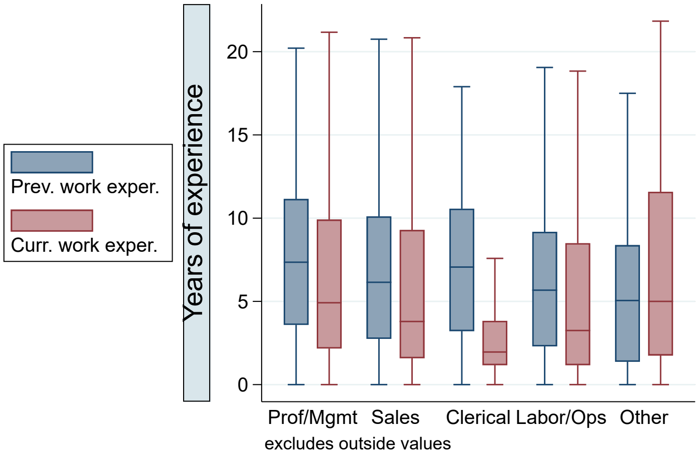
Looking at previous and current work experience over occupations, we can use the `ytitle()` option to add a title to the y axis. See Options : Axis titles (section 8.4) and `help axis_title_options` for more details. *Uses nlsw.dta*



```
graph box prev_exp tenure, nooutsides over(occ5)
ytitle("Years of experience", size(vlarge) box bexpand)
scheme(vg_lgndc)
```

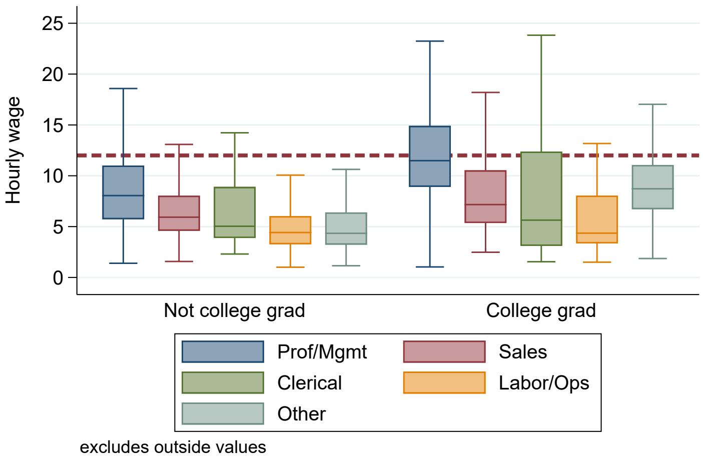
The text specified within the `ytitle()` option is considered a textbox, so we can use textbox options to control the look of the title. This example makes the title very large, surrounds it with a box, and uses the `bexpand` (box expand) option to stretch the box to fill the width of the plot area. See

Options : Textboxes (section 8.11) for more examples of how to use textbox options to control the display of text (including how to size text using points, inches, and centimeters). *Uses nlsw.dta*



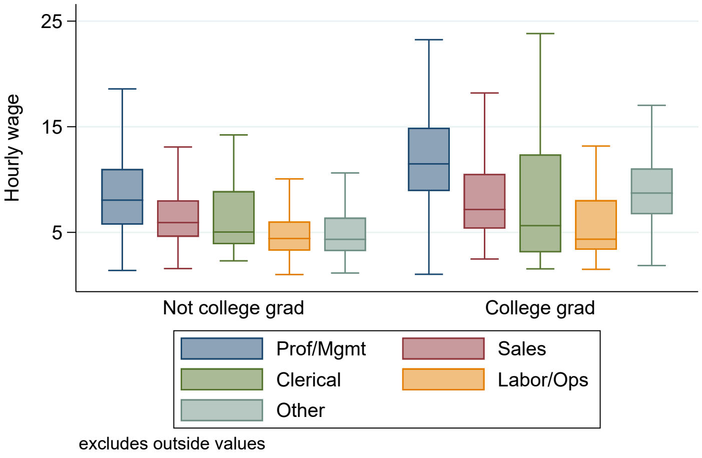
```
graph box wage, nooutsides over(occ5) over(collgrad) asyvar
yline(12, lwidth(thick) lcolor(maroon) lpattern(dash))
```

Here we use the `yline()` option to add a thick, maroon, dashed line to the points in the graph where wages equal 12. We would still use `yline()`, even if we used `graph hbox`, placing the *y* axis at the bottom. *Uses nlsw.dta*

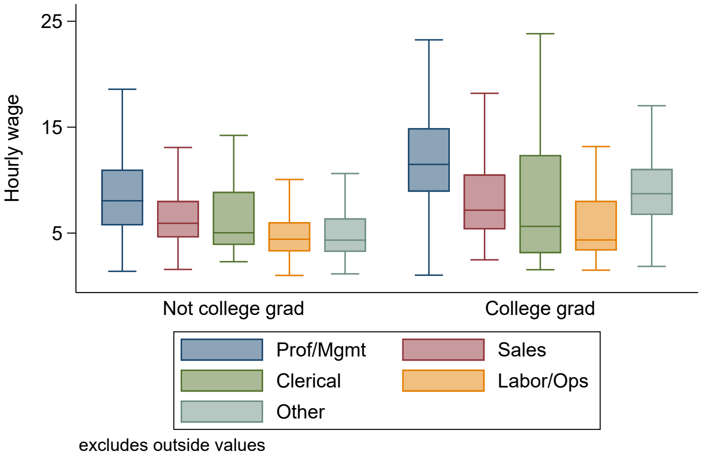


```
graph box wage, nooutsides over(occ5) over(collgrad) asyvar
ylabel(5(10)25)
```

We use the `ylabel()` option to label the *y* axis going from 5 to 25, incrementing by 10. The *y* axis still starts at 0, but we could override this by using the `exclude0` option. See **Options : Axis labels** (section 8.5) for more details. *Uses nlsw.dta*



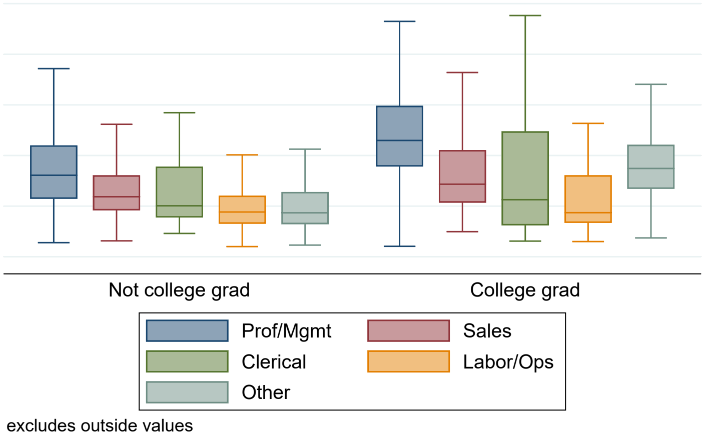
```
graph box wage, nooutsides over(occ5) over(collgrad) asyvar
ylabel(5(10)25, nogrid)
```



We use the `nogrid` option to suppress the display of the grid for the y axis. (With box plots, there is never a grid with respect to the x axis.) If the grid were absent and you wanted to include it, you could add the `grid` option. For more details, see Options : Axis labels (section 8.5). *Uses nlsw.dta*

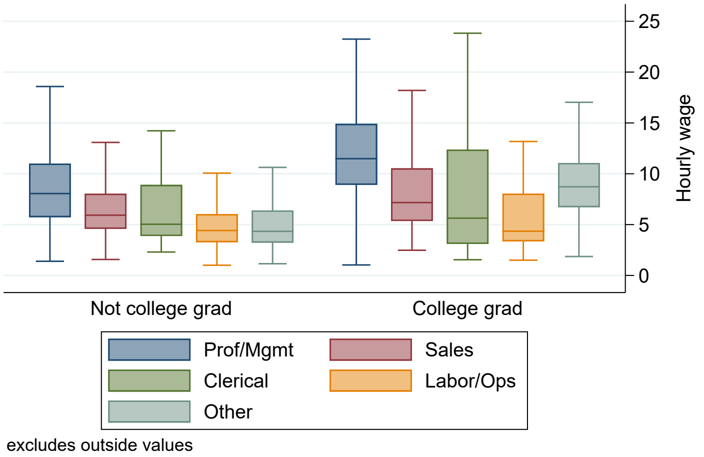
```
graph box wage, nooutsides over(occ5) over(collgrad) asyvar
yscale(off)
```

We can use `yscale(off)` to turn off the y axis. See Options : Axis scales (section 8.6) and `help axis_scale_options` for more details. Disregard any references to `xscale()` because this option is not valid when using `graph box`. *Uses nlsw.dta*



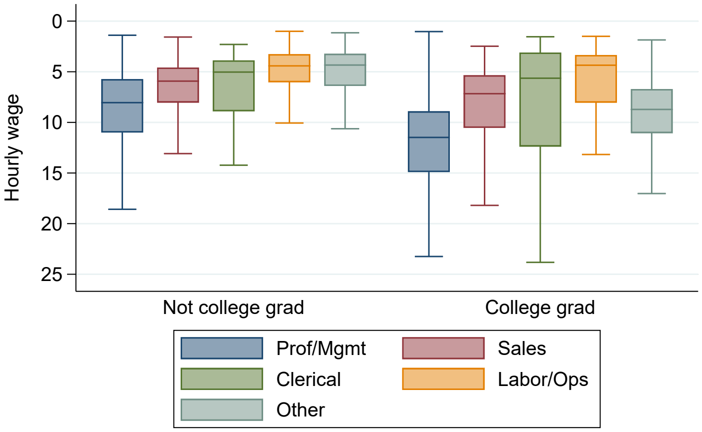
```
graph box wage, nooutsides over(occ5) over(collgrad) asyvar
yalternate
```

We can put the *y* axis on the opposite side, which here is on the right side of the graph, by using the `yalternate` option. *Uses nlsw.dta*



```
graph box wage, nooutsides over(occ5) over(collgrad) asyvar
yreverse
```

You can reverse the direction of the *y* axis, in effect turning your boxes upside down, with the `yreverse` option. *Uses nlsw.dta*



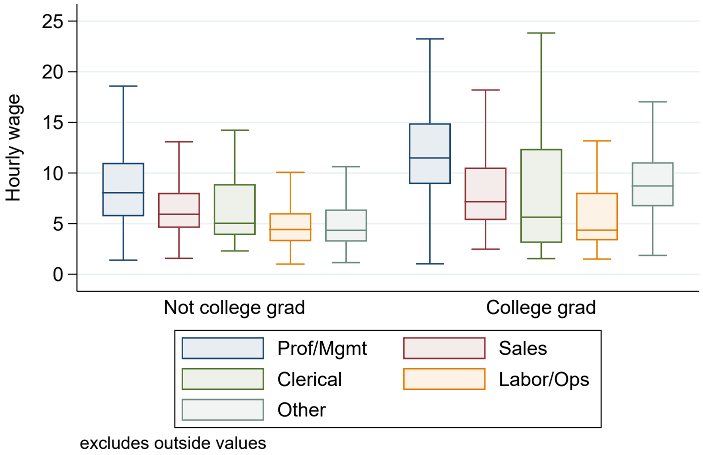
excludes outside values

5.7 Changing the look of boxes

This section shows how we can control the color of the boxes and the characteristics of the line outlining the boxes. We will first look at options that control the overall intensity of the color for all the boxes and then show how you can control the color of each box.

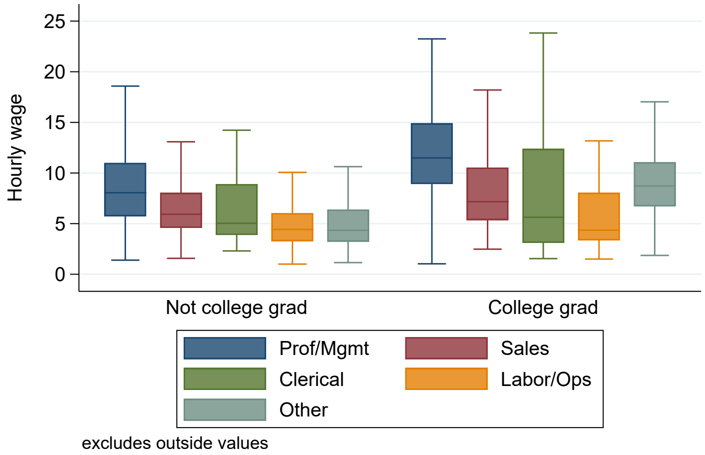
```
graph box wage, over(occ5) over(collgrad) asyvars nooutsides
intensity(10)
```

The `intensity()` option controls the intensity of the color within the boxes. Here we request that the color be displayed with 10% intensity (contrasted with the default, which is 50%). *Uses nlsw.dta*



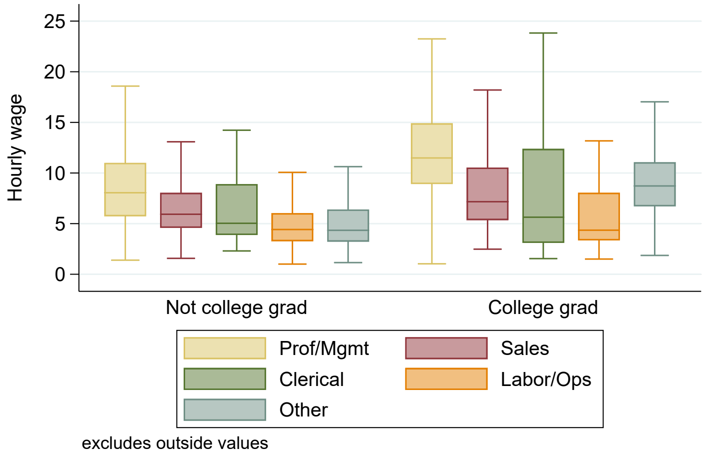
```
graph box wage, over(occ5) over(collgrad) asyvars nooutsides
intensity(80)
```

In this example, we use the `intensity()` option to make the intensity of the colors 80%. *Uses nlsw.dta*



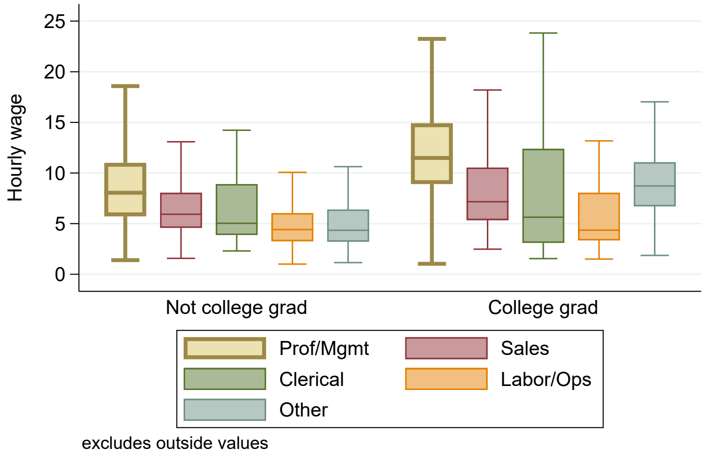
```
graph box wage, over(occ5) over(collgrad) asyvars nooutsides
box(1, bcolor(sand))
```

Here we add `box(1, bcolor(sand))` to specify that the box color for the first bar be sand. See **Styles : Colors** (section 10.2) for more information about the available colors. *Uses nlsw.dta*



```
graph box wage, over(occ5) over(collgrad) asyvars nooutsides
box(1, bcolor(sand) blcolor(brown) blwidth(thick))
```

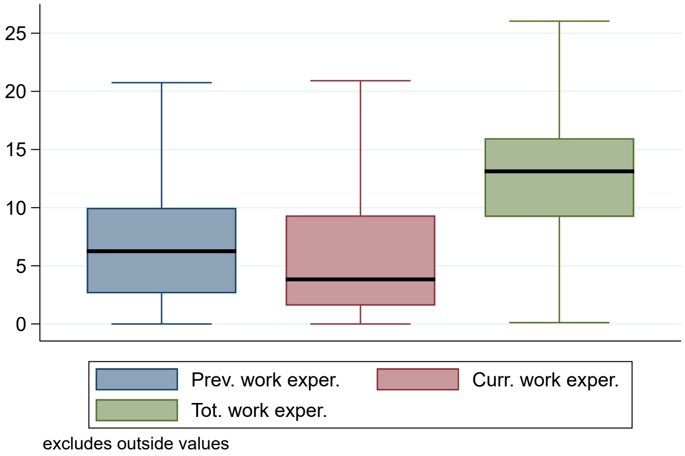
We add the `blcolor()` (box line color) and `blwidth()` (box line width) options to make the outline for the first box brown and thick. *Uses nlsw.dta*



Now let's consider options that allow us to control the display of the median, whiskers, caps, and outside markers.

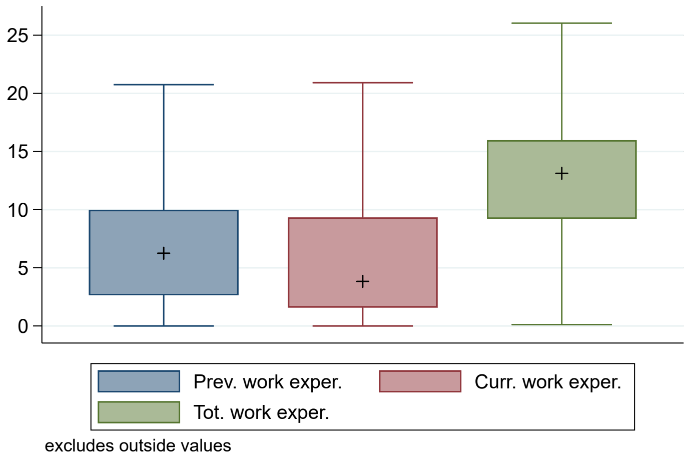
```
graph box prev_exp tenure ttl_exp, nooutsides medtype(cline)
medline(lwidth(thick) lcolor(black))
```

The `medtype(cline)` option sets the median type to be a custom line. We then customize the median line by using the `medline()` option to specify that the line width be thick and the line color be black. *Uses nlsw.dta*



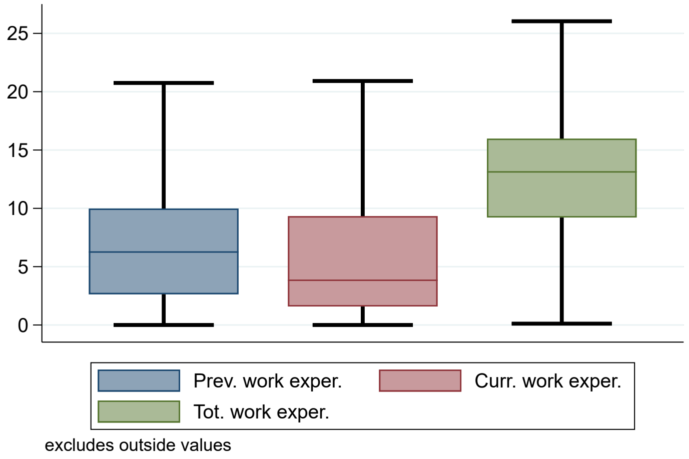
```
graph box prev_exp tenure ttl_exp, nooutsides medtype(marker)
medmarker(msymbol(+) msize(large))
```

The `medtype(marker)` option specifies that we want to use a marker symbol to label the median, and the `medmarker()` option controls the display of the median marker. Here we make the marker symbol a plus sign and the marker size large. *Uses nlsw.dta*



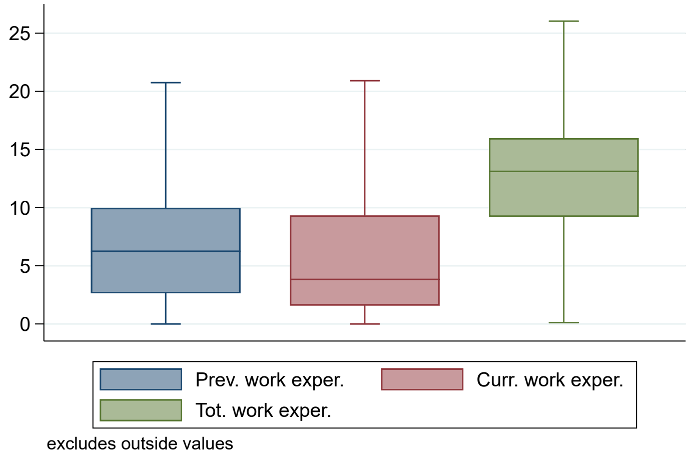
```
graph box prev_exp tenure ttl_exp, nooutsides cwhiskers
lines(lwidth(thick) lcolor(black))
```

To customize the whiskers, we need to specify the `cwhiskers` (customize whiskers) option, and then we can add the `lines()` option to specify how we want the whiskers customized. In this example, we make the whiskers thick and black. *Uses nlsw.dta*



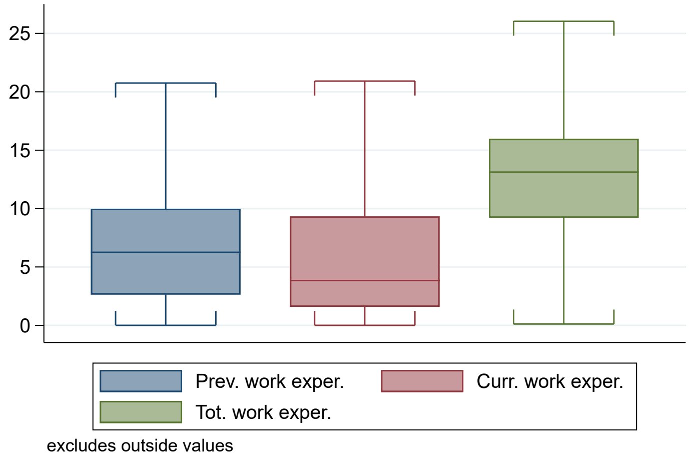
```
graph box prev_exp tenure ttl_exp, nooutsides alsize(20)
```

The `alsize()` (adjacent line size) option allows us to control the size (width) of the adjacent line. By default, the adjacent line is 67% of the width of the box. Here we make the adjacent line much smaller, 20% of the width of the box. *Uses nlsw.dta*



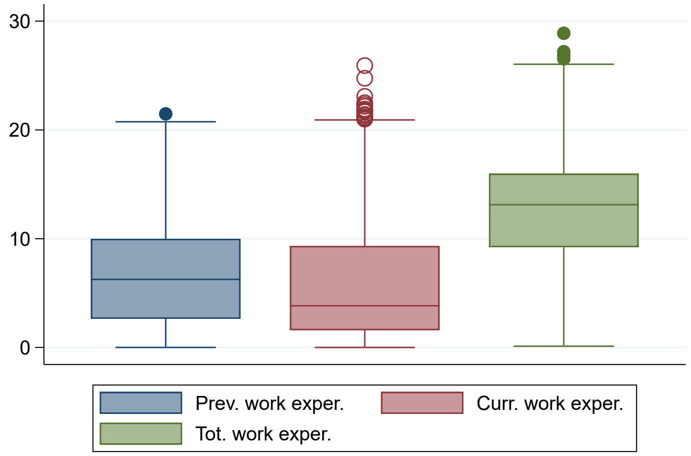
```
graph box prev_exp tenure ttl_exp, nooutsides capsize(5)
```

The `capsize()` option allows us to specify the size of the caps (if any) on the adjacent line. The default value is 0, meaning that no cap is displayed. In this example, we add a small cap to the adjacent line. *Uses nlsw.dta*



```
graph box prev_exp tenure ttl_exp, marker(2, msymbol(Oh)
msize(vlarge))
```

The `marker()` option allows us to control the markers used to display the outside values. We can control this separately for each y variable. Here we make the outside value for `tenure` display as large, hollow circles. *Uses nlsw.dta*

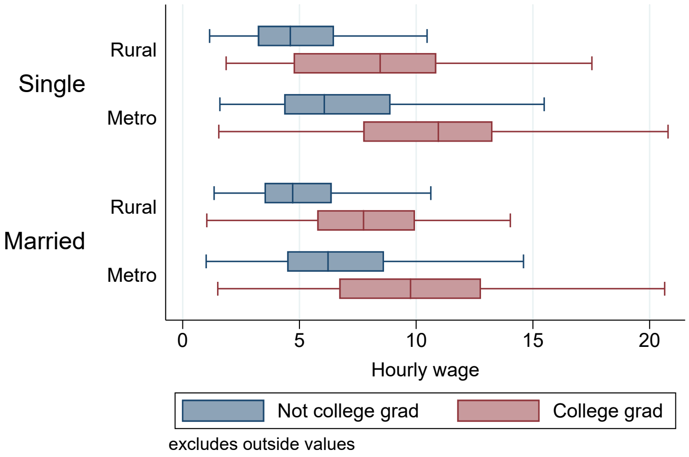

30
20
10
0
Prev. work exper.
Curr. work exper.
Tot. work exper.

5.8 Graphing by groups

This section discusses the use of the `by()` option in combination with `graph box`. Normally, you would use the `over()` option instead of the `by()` option, but sometimes the `by()` option is either necessary or more advantageous. For example, a `by()` option is useful if you exceed the maximum number of `over()` options (three if you have one y variable or two if you have multiple y variables). Thus the `by()` option allows you to break down your data by more categorical variables. Also, `by()` gives you more flexibility in the placement of the separate panels. For more information about the `by()` option, see Options : By (section 8.8); for more information about the `over()` option, see Box : Yvars and over (section 5.1).

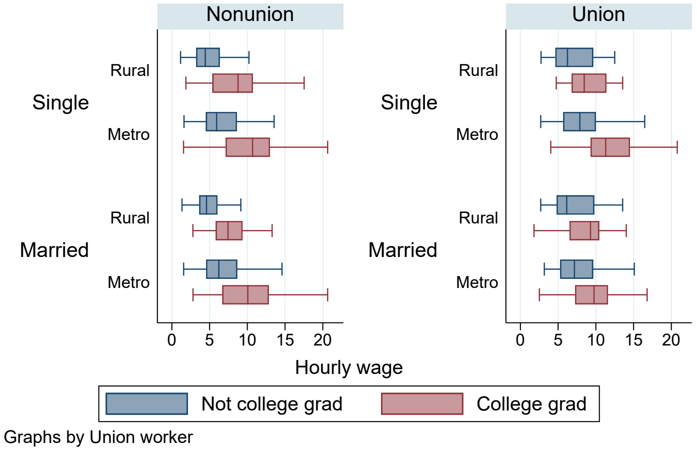
```
graph hbox wage, nooutsides over(collgrad) over(urban2)
over(married)
```

Consider this box graph, which breaks down wages by three categorical variables. If we wanted to further break down this by another categorical variable, we could not use another `over()` option because we can have a maximum of three `over()` options with one y variable. *Uses nlsw.dta*



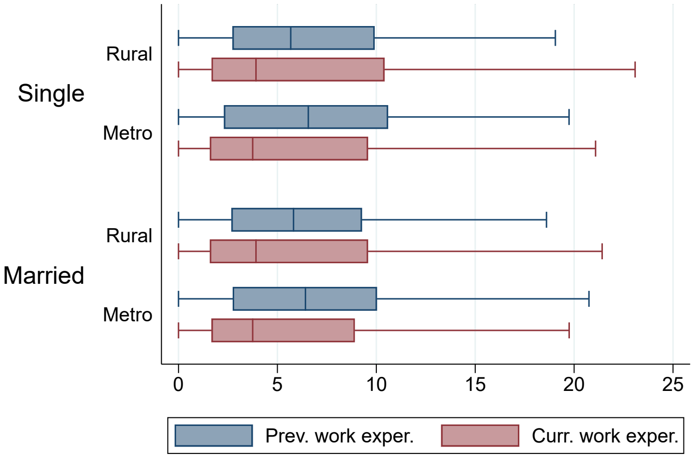
```
graph hbox wage, nooutsides note("") over(collgrad)
over(urban2) over(married) by(union)
```

If we want to further break down `prev_exp` by `union`, we can use the `by(union)` option. We also add the `note("")` option to suppress the note "excludes outside values". *Uses nlsw.dta*



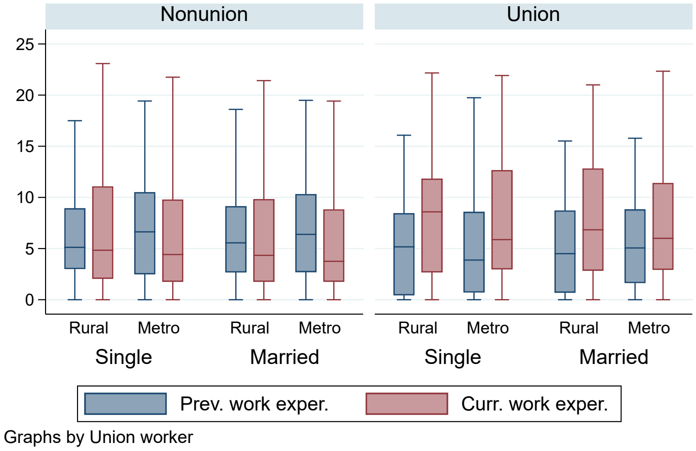
```
graph hbox prev_exp tenure, nooutsides note("") over(urban2)
over(married)
```

Consider this box graph with multiple y variables breaking down them by two categorical variables using two `over()` options. When we have multiple y variables, we can have a maximum of two `over()` options. *Uses nlsw.dta*



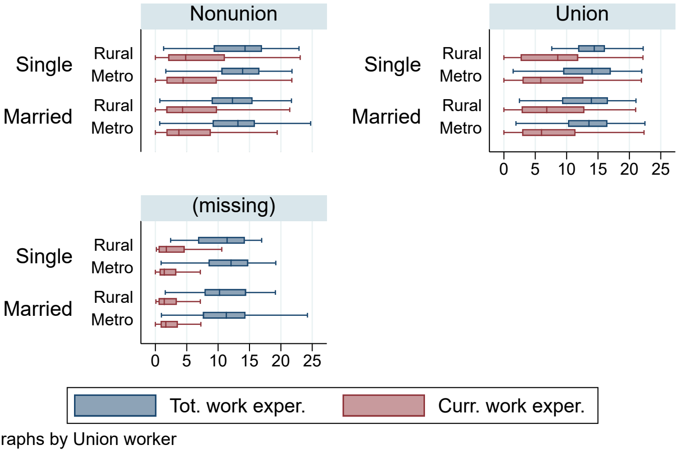
```
graph box prev_exp tenure, nooutsides note("") over(urban2)
over(married) by(union)
```

We can add the `by(union)` option to further break down these `by` plots by union status. We can include multiple variables within `by()`, although this can make some small graphs. *Uses nlsw.dta*



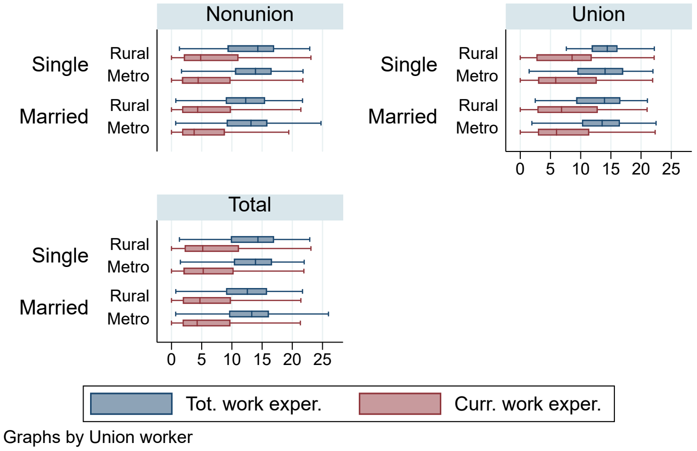
```
graph hbox ttl_exp tenure, nooutsides note("") over(urban2)
over(married) by(union, missing)
```

We can use the `missing` option to include a panel for the missing values of `union`. *Uses nlsw.dta*



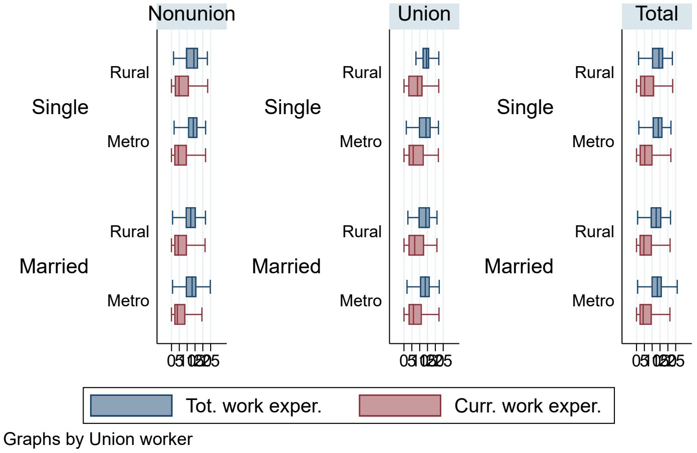
```
graph hbox ttl_exp tenure, nooutsides note("") over(urban2)
over(married) by(union, total)
```

We can add the `total` option to include a panel for all observations. *Uses nlsw.dta*



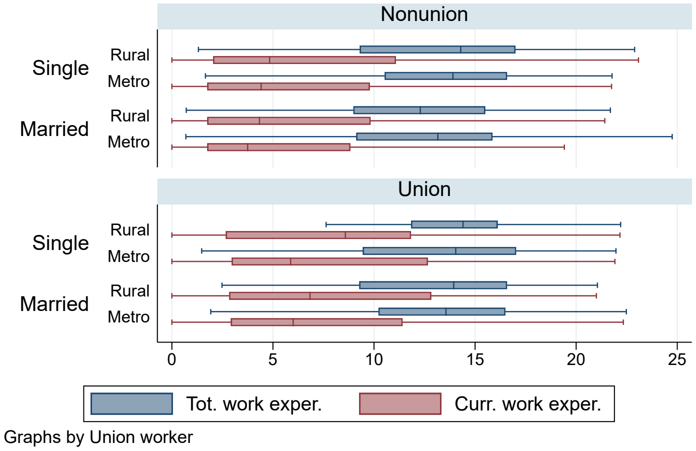
```
graph hbox ttl_exp tenure, nooutsides note("") over(urban2)
over(married) by(union, total rows(1))
```

We can use the `rows(1)` option to show the multiple graphs in one row. *Uses nlsw.dta*



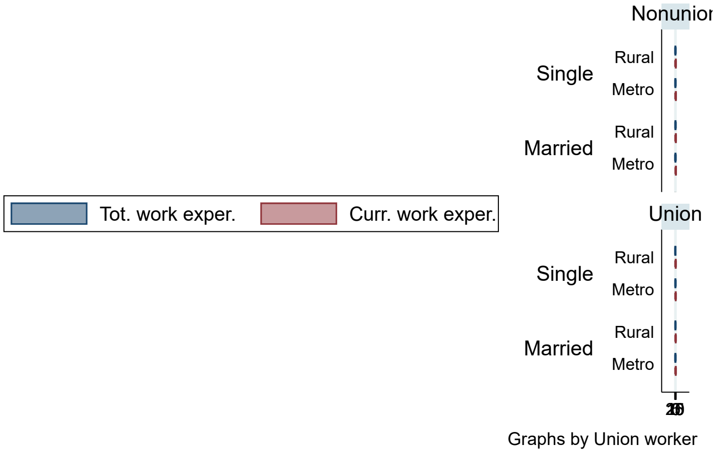
```
graph hbox ttl_exp tenure, nooutsides note("") over(urban2)
over(married) by(union, cols(1))
```

Here we omit the `total` option and use the `cols(1)` option to show both graphs in one column. *Uses nlsw.dta*



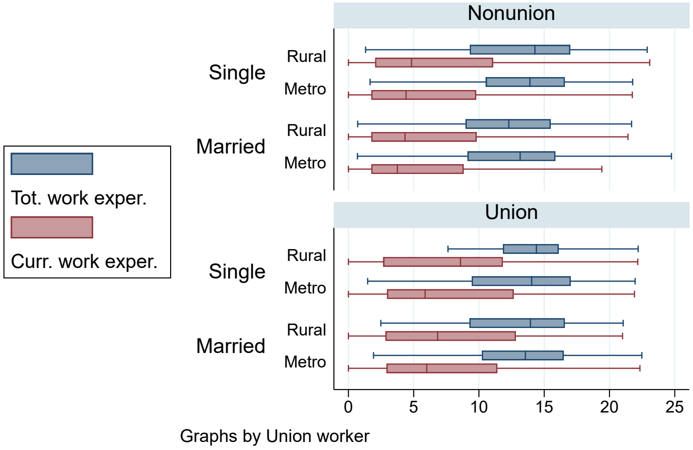
```
graph hbox ttl_exp tenure, nooutsides note("") over(urban2)
over(married) by(union, cols(1) legend(position(9)))
```

We use the `legend(position(9))` option within the `by()` option to put the legend at nine o'clock. *Uses nlsw.dta*



```
graph hbox ttl_exp tenure, nooutsides note("") over(urban2)
over(married) by(union, cols(1) legend(position(9)))
legend(cols(1) stack)
```

We now add the `legend(cols(1) stack)` option to make the legend one stacked column. *Uses nlsw.dta*


Tot. work exper.
Curr. work exper.
Nonunion
Single
Rural
Metro
Married
Rural
Metro
Union
Single
Rural
Metro
Married
Rural
Metro
0
5
10
15
20
25
Graphs by Union worker

Chapter 6
Dot plots

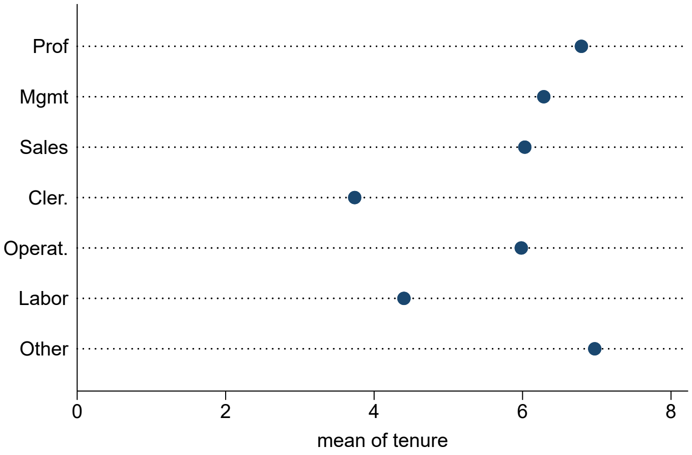
This chapter illustrates how to create dot plots using the `graph dot` command. The first section shows how to create dot plots as a function of one or more categorical variables via the `over()` option, and it shows how to create graphs of multiple y variables. Next I illustrate ways that you can control the gaps among dots on the categorical axis. The next section shows how to control the sorting of dots, followed by a section that illustrates how to customize the labels of the categorical axis. Next I show how you can customize the display of the legend, including how to modify the labels and position the legend. The following section shows how you can customize the y-axis title and the y-axis labels and even hide the y axis altogether. In the next section, I illustrate options that control the look of the lines and dots that form the dot plot. The last section illustrates how to use the `by()` option to create multiple graphs as a function of the levels of the *by-variable* specified in the `by()` option. For this entire chapter, we will use the `nlsw` data file.

6.1 Specifying variables and groups

This section introduces the use of dot plots. It shows how we can use the `over()` option for displaying dot plots by one or more grouping variables. It then shows how we can specify one or more y variables in a plot and control the summary statistic used for collapsing the y variable(s). See the *group_options* table in `help graph dot` for more details.

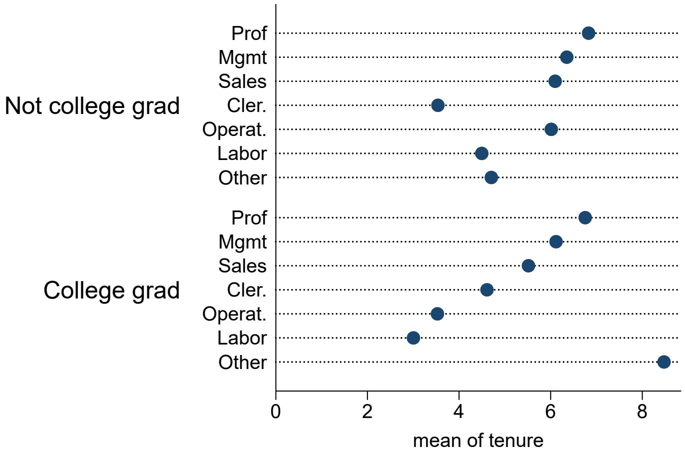
```
graph dot tenure, over(occ7)
```

Here we use the `over()` option to show the average current work experience broken down by occupation. By default, the y variable (`tenure`) is placed on the bottom axis and is considered the y axis. Likewise, the levels of `occ7` are placed on the left axis and are considered to form the x axis, or categorical axis. *Uses nlsw.dta*

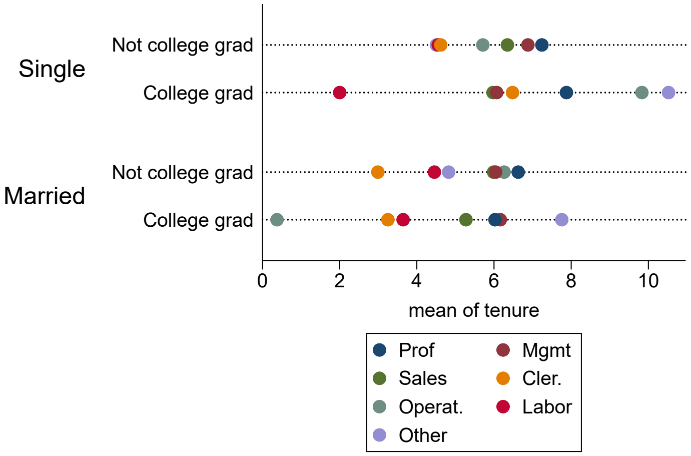


```
graph dot tenure, over(occ7) over(collgrad)
```

Here we use a second `over()` option to show the mean of work experience broken down by occupation and whether one graduated college. *Uses nlsw.dta*



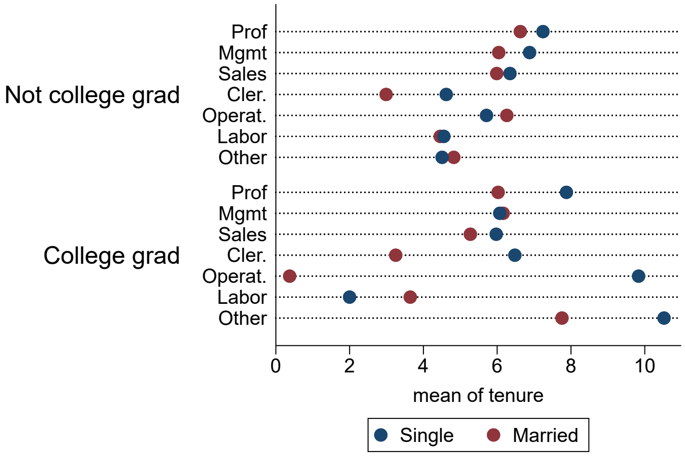
```
graph dot tenure, over(occ7) over(collgrad) over(married)
```



We can add a third `over()` option to further break down `tenure` by whether one is married. The first `over()` variable (`occ7`) is now treated as multiple y variables. When we use three `over()` options, the first variable is then treated as multiple y variables, as though we had specified the `asyvars` option. This graph can be difficult to read with `occ7` forming the multiple y variables. *Uses nlsw.dta*

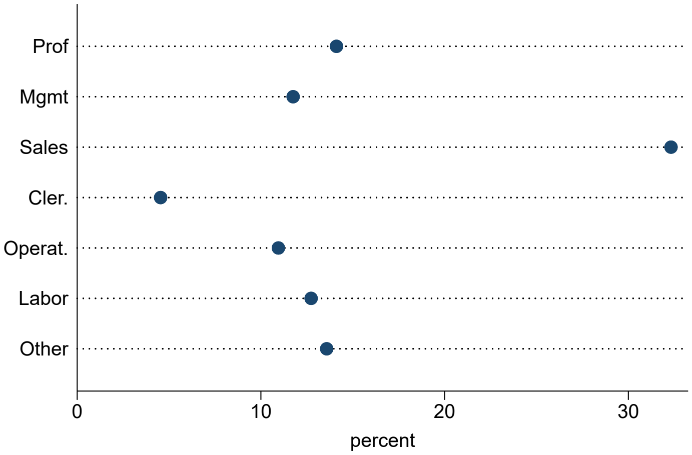
```
graph dot tenure, over(married) over(occ7) over(collgrad)
```

This graph shows the same data as the last one, except that we have switched the order of the `over()` options, making `over(married)` come first and thus forming the multiple y variables. This might be easier to read than the previous graph. *Uses nlsw.dta*



The prior examples graphed average work experience (`tenure`) broken down by grouping variables (such as occupation or whether one graduated from college). The following examples illustrate how you can construct dot plots that depict the percentage of observations that fall within each category (such as occupation or whether one graduated from college).

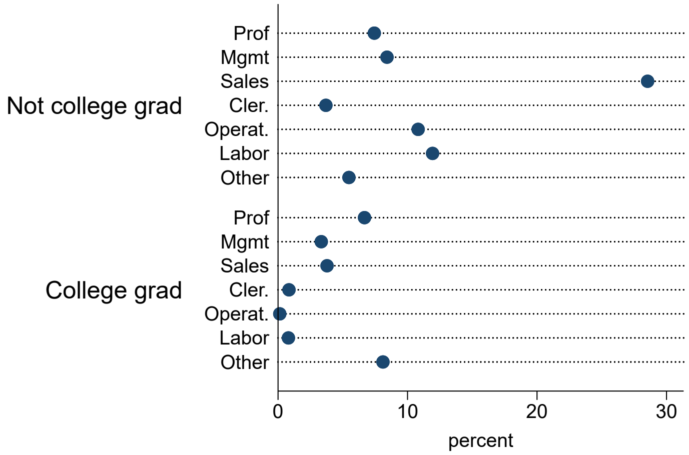
```
graph dot (percent), over(occ7)
```



This dot plot shows the percentage of observations that belong to each of the seven occupations defined by `occ7`. For example, over 30% of workers in this sample worked in sales. By specifying the `over(occ7)` option, we break down the results by `occ7`, and by specifying `(percent)` after `graph dot`, we indicate that each dot represents the percentage of all observations within each level of `occ7`. *Uses nlsw.dta*

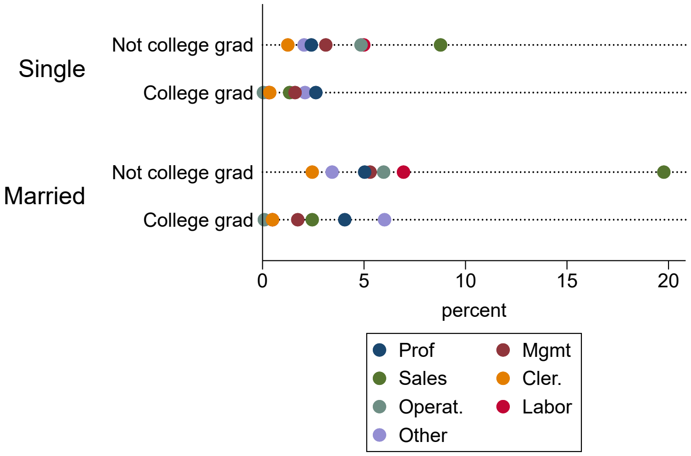
```
graph dot (percent), over(occ7) over(collgrad)
```

This example includes two `over()` variables. The percentages graphed mirror the percentages we would obtain if we performed a cross-tabulation of `occ7` by `collgrad` and computed the cell percentages. For example, about 29% of all workers were noncollege graduates working in sales. *Uses nlsw.dta*



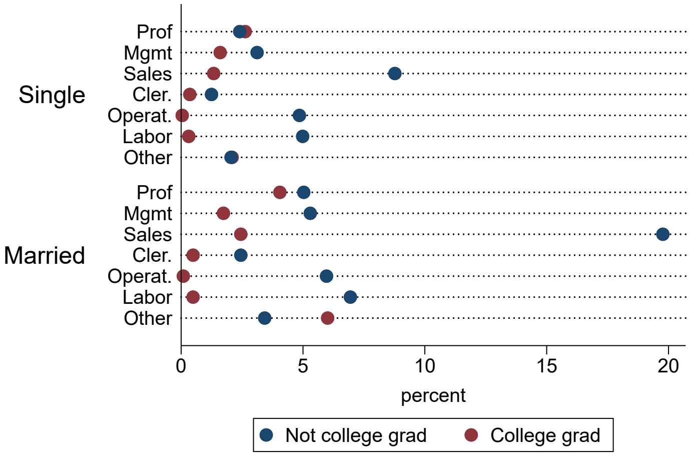
```
graph dot (percent), over(occ7) over(collgrad) over(married)
```

This example includes three `over()` variables. The dots represent the cell percentages from a three-way cross-tabulation of `occ7` by `collgrad` by `married`. For example, about 20% of all workers are married, noncollege graduates, working in sales. Note that because of the inclusion of the third `over()` option, the levels of the first `over()` variable are displayed using different colored dots, labeled using a legend. *Uses nlsw.dta*



```
graph dot (percent), over(collgrad) over(occ7) over(married)
```

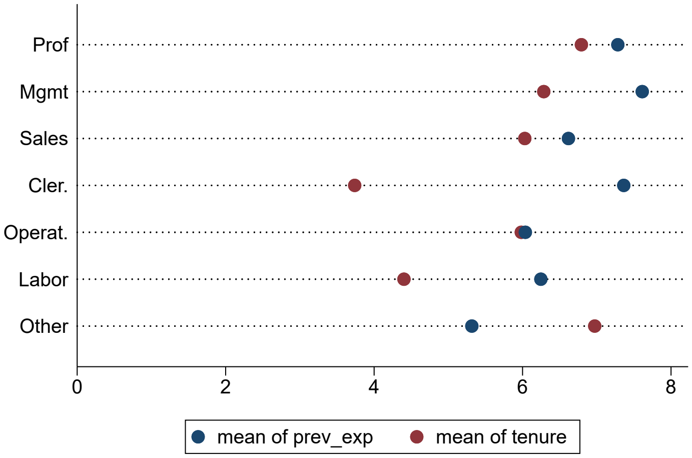
This example is like the previous example, except that the order of the `over()` variables was changed. In this example, the blue dots represent the noncollege graduates, and the red dots represent the college graduates. We see that about 20% of workers are married noncollege graduates working in sales, while about 2% are married, college graduates working in sales. *Uses nlsw.dta*



Let's now consider examples with multiple y variables.

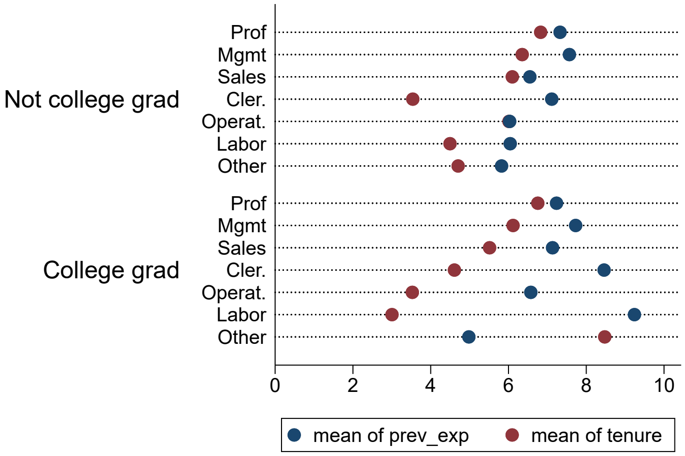
```
graph dot prev_exp tenure, over(occ7)
```

This graph shows the average previous experience and average current tenure broken down by occupation. Although we do not need to use the `over()` option, omitting it may make a fairly boring graph. *Uses nlsw.dta*



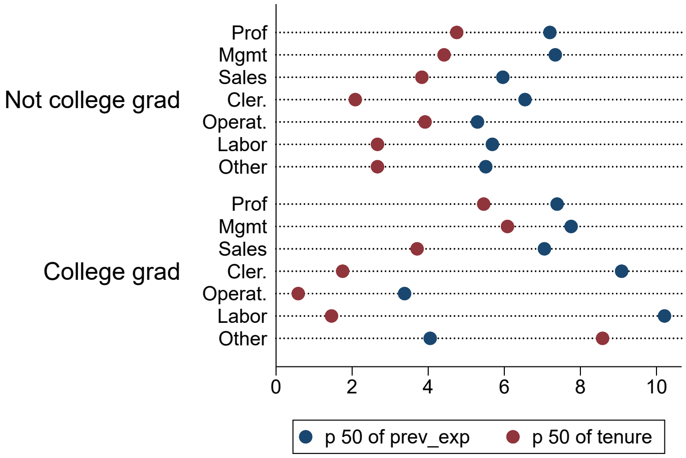
```
graph dot prev_exp tenure, over(occ7) over(collgrad)
```

This graph also shows whether one is a college graduate as another grouping level. The command has multiple y variables, so we cannot include another `over()` option because dot plots support three levels of nesting and the multiple y variables account for a level. *Uses nlsw.dta*



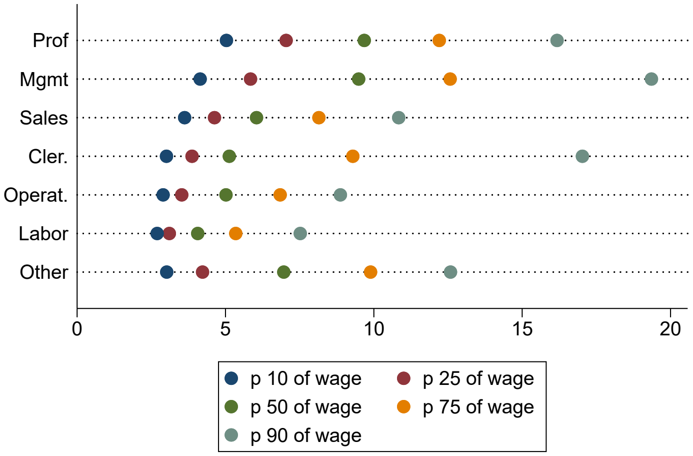
```
graph dot (median) prev_exp tenure, over(occ7) over(collgrad)
```

So far, all the examples we have seen have graphed the mean of y variable(s). Here we preface the y variables with `(median)`, plotting the median for each y variable. *Uses nlsw.dta*



```
graph dot (p10) wage (p25) wage (p50) wage (p75) wage (p90)
wage, over(occ7)
```

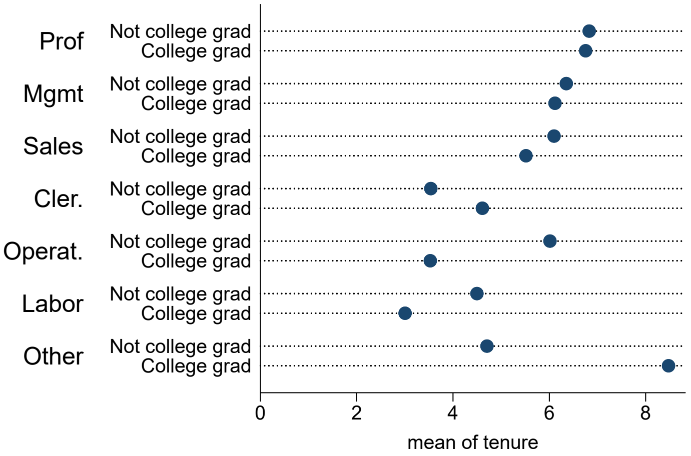
We can request different statistics for the same variable, such as here, which shows the 10th, 25th, 50th, 75th, and 90th percentiles of wages broken down by occupation. *Uses nlsw.dta*



Now let's consider options that we can use with the `over()` option to customize the behavior of the graphs. We will see how to treat the levels of the first `over()` option as though they were multiple y variables. We can also request that missing values for the levels of the `over()` variables be displayed, and we can suppress empty categories when multiple `over()` options are used.

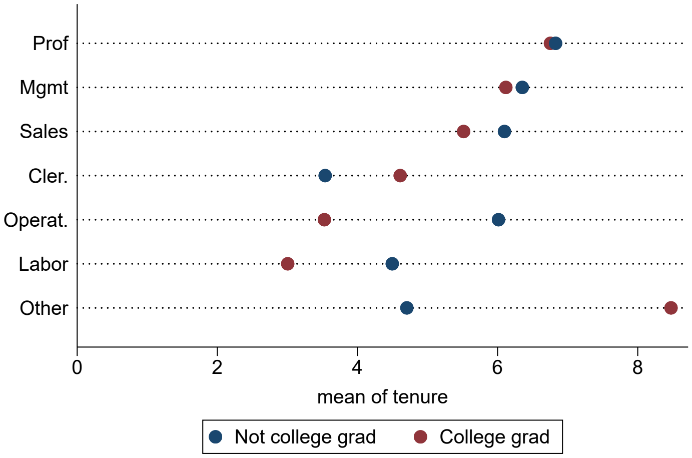
```
graph dot tenure, over(collgrad) over(occ7)
```

Consider this graph showing the average current work experience broken down by whether one is a college graduate and by occupation. *Uses nlsw.dta*



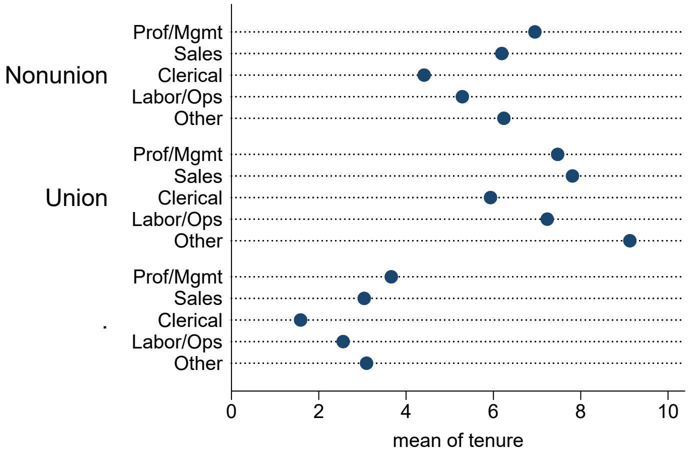
```
graph dot tenure, over(collgrad) over(occ7) asyvars
```

With the `asyvars` option, the first `over()` variable (`collgrad`) is graphed as if there were two y variables. The two levels of `collgrad` are shown as different markers on the same line, and they are labeled using the legend. *Uses nlsw.dta*



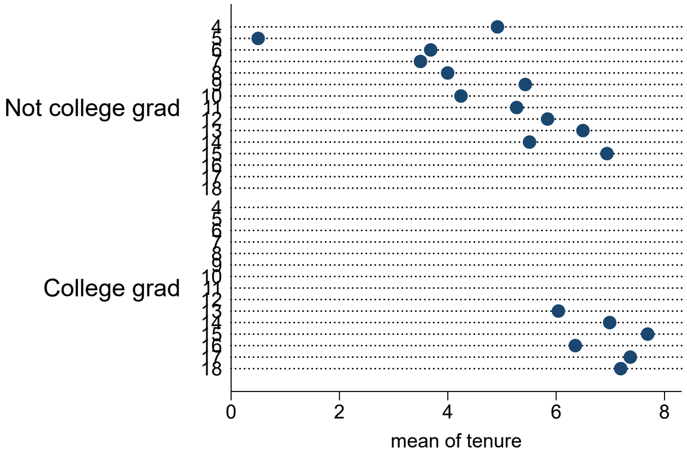
```
graph dot tenure, over(occ5) over(union) missing
```

Consider this graph in which we use the `over()` option to show `tenure` broken down by `occ5` and `union`. By including the `missing` option, we see the category for those who are missing on the `union` variable, shown as the third group labeled with a dot. See Dot : Cat axis (section 6.4) for examples showing how we could change the label (.) to something more meaningful, for example, “Missing”. *Uses nlsw.dta*



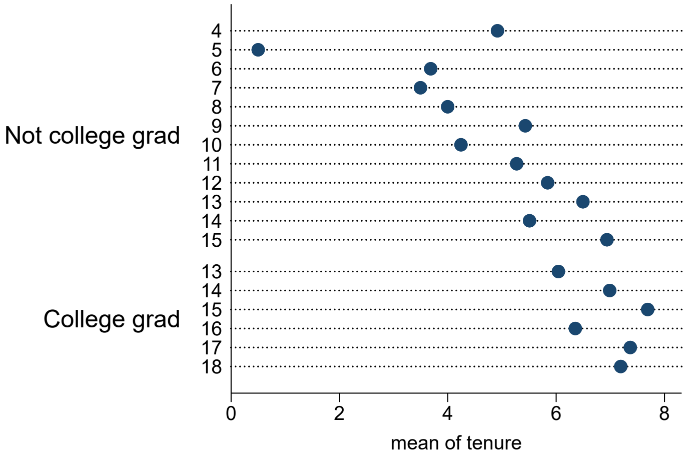
```
graph dot tenure, over(grade) over(collgrad)
```

Consider this dot plot, which breaks down `tenure` by two variables: the last grade that one completed and whether one is a college graduate. By default, Stata shows all possible combinations for these two variables. Usually, all combinations are possible, but not here, and including them has caused the labels for grade to overlap. *Uses nlsw.dta*



```
graph dot tenure, over(grade) over(collgrad) nofill
```

If we want to display only the combinations of the `over()` variables that exist in the data, we can use the `nofill` option. *Uses nlsw.dta*



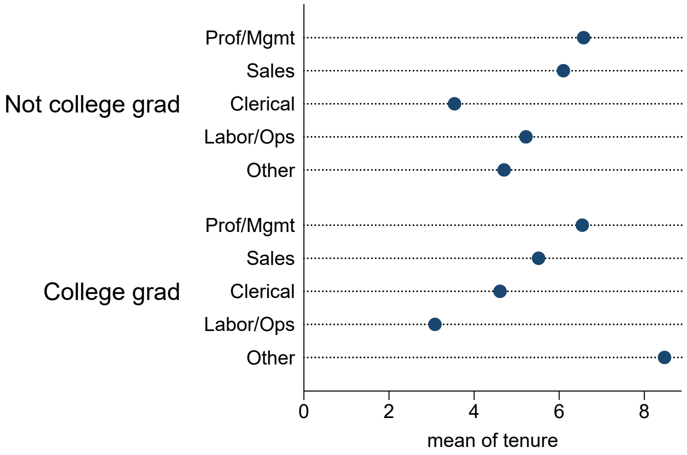
6.2 Options for controlling gaps between dots

This section considers some of the options that we can use for controlling the spacing between the markers. For more information on the options covered in this section, see the *over_subopts* table in `help graph dot`.

When dealing with bar and box plots, the spacing was measured for the width of a bar or box. The gap could be increased or decreased a fraction of the width of a bar/box. With a dot plot, the same concept applies except the width is measured not in terms of the dot itself, but in terms of the width of the entire line the dot occupies.

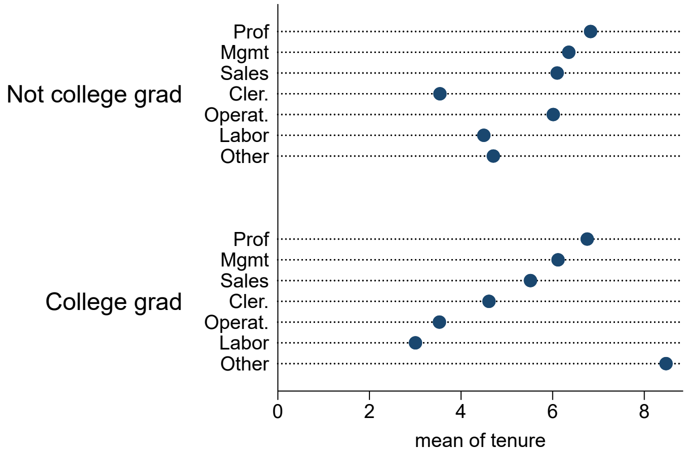
```
graph dot tenure, over(occ5) over(collgrad)
```

Consider this graph in which we show a dot plot of `tenure` broken down by `occ5` and `collgrad`. Note the width of each line for each occupation. *Uses nlsw.dta*



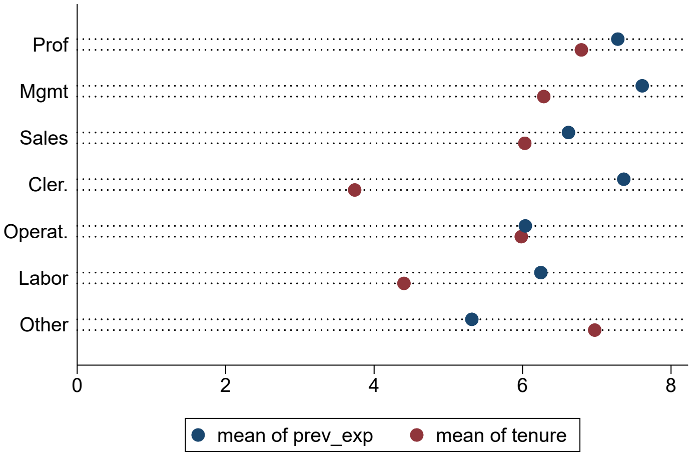
```
graph dot tenure, over(occ7) over(collgrad, gap(300))
```

Suppose that we wanted to make the gap between the levels of `collgrad` larger. Here we use the `gap(300)` option to make this gap 300% (three times) the width of an individual line. *Uses nlsw.dta*



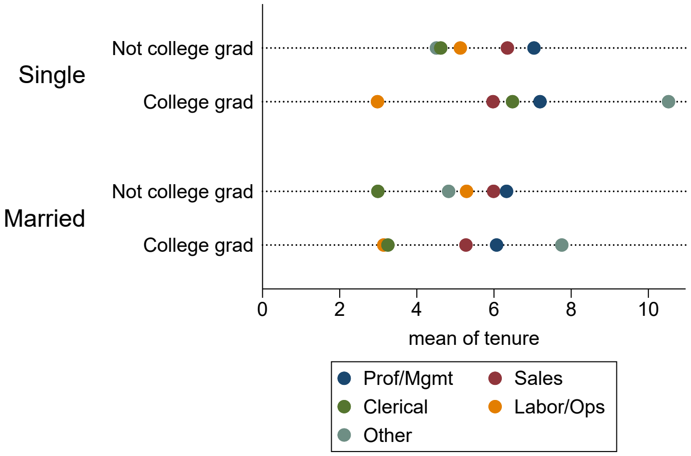
```
graph dot prev_exp tenure, over(occ7) linegap(30)
```

We use the `linegap()` option to display the y variables on different lines and specify the gap between these lines. The gap is 30% of the width of the entire line. The default value is 0, meaning that all y variables are displayed on the same line. *Uses nlsw.dta*



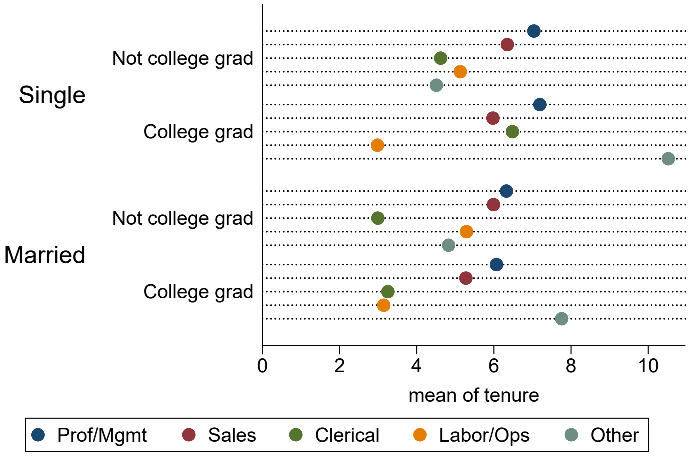
```
graph dot tenure, over(occ5) over(collgrad) over(married)
```

Consider this graph. Because we have used three `over()` options, the levels of the first `over()` variable are displayed as though they were different y variables. We may want to use the `linegap()` option to display the different y variables on different lines to make the graph more readable; see the next example. *Uses nlsw.dta*



```
graph dot tenure, over(occ5) over(collgrad) over(married)
linegap(70) legend(rows(1) span)
```

This example is similar to the previous one, but we have added the `linegap(70)` option to make the levels of `occ5` display on separate lines, making the results more readable. We have also added a `legend()` option to make the legend display in one line and span the width of the graph. *Uses nlsw.dta*

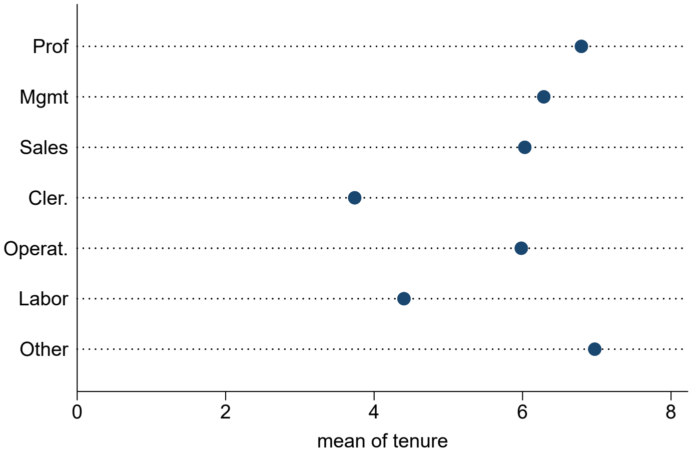


6.3 Options for sorting dots

This section considers some of the options that we can use to change the order in which the markers are sorted.

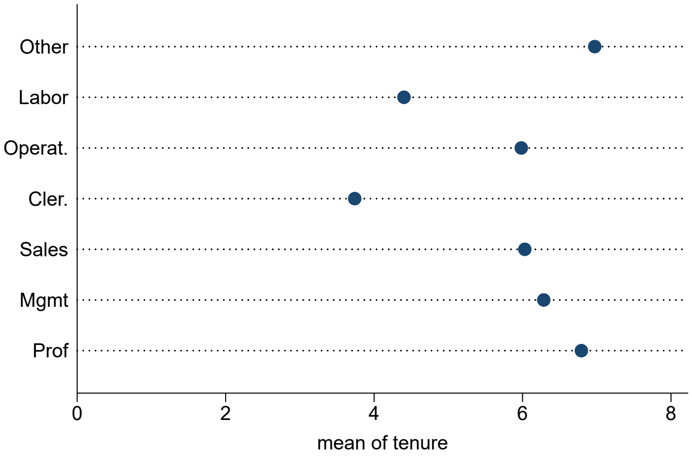
```
graph dot tenure, over(occ7)
```

Consider this graph showing `tenure` broken down by the seven levels of occupation. The markers are ordered by levels of `occ7`, going from 1 to 7. *Uses nlsw.dta*



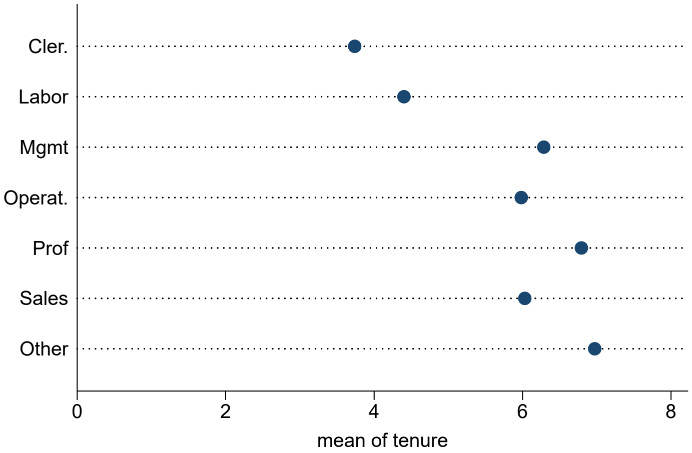
```
graph dot tenure, over(occ7, descending)
```

The `descending` option switches the order of the markers. They are still ordered according to the seven levels of occupation, but the markers are ordered from 7 to 1. *Uses nlsw.dta*



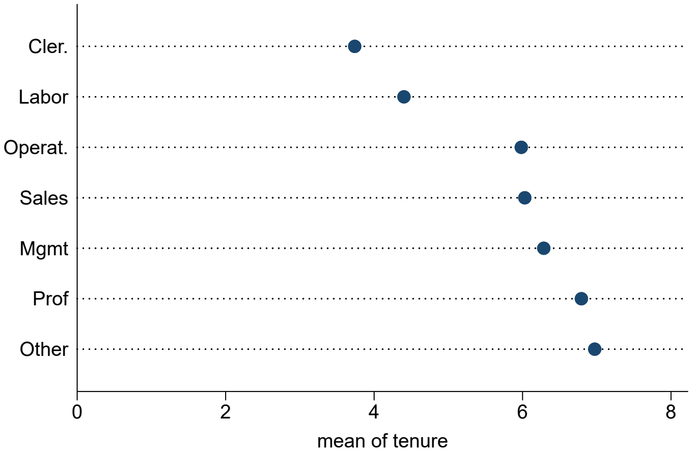
```
graph dot tenure, over(occ7, sort(occ7alpha))
```

We might want to put these markers in alphabetical order but with *Other* appearing last. We can do this by recoding `occ7` into a new variable (say, `occ7alpha`) such that as `occ7alpha` goes from 1 to 7, the occupations are alphabetical. We recoded `occ7` with these assignments: $4 = 1$, $6 = 2$, $2 = 3$, $5 = 4$, $1 = 5$, $3 = 6$, and $7 = 7$; see `help recode`. Then the `sort(occ7alpha)` option alphabetizes the markers but with *Other* still appearing last. *Uses nlsw.dta*



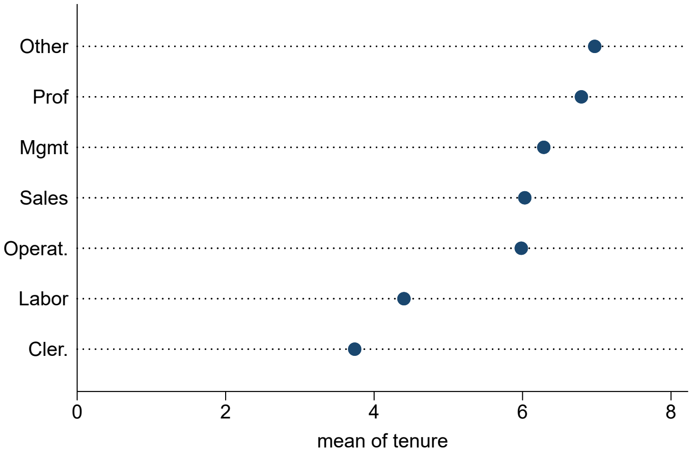
```
graph dot tenure, over(occ7, sort(1))
```

Here we sort the variables on the mean of `tenure`, yielding markers with means in ascending order. The `sort(1)` option sorts the markers according to the mean of the first y variable, the mean of `tenure`. *Uses nlsw.dta*



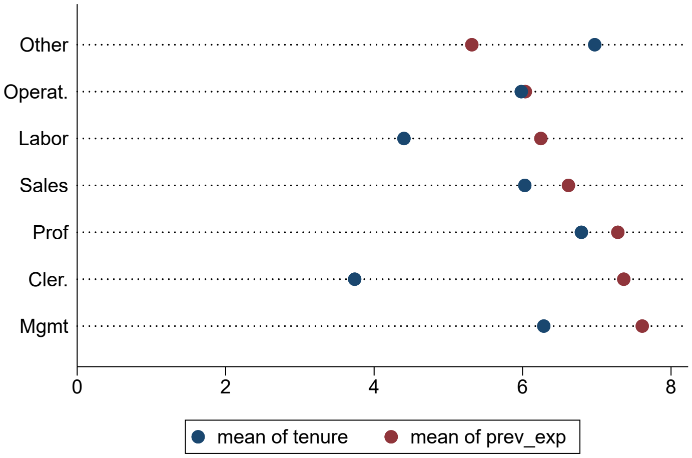
```
graph dot tenure, over(occ7, sort(1) descending)
```

Adding the `descending` option yields markers in descending order, going from highest mean `tenure` to lowest mean `tenure`. *Uses nlsw.dta*



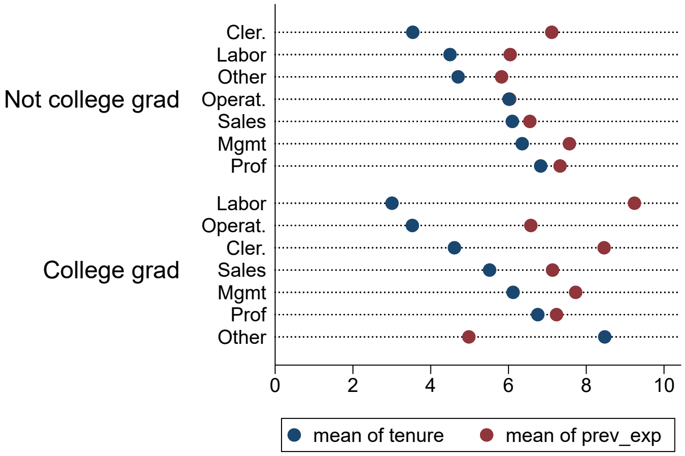
```
graph dot tenure prev_exp, over(occ7, sort(2))
```

Adding a second y variable and changing `sort(1)` to `sort(2)` sorts the markers according to the second y variable, the mean of `prev_exp`. *Uses nlsw.dta*



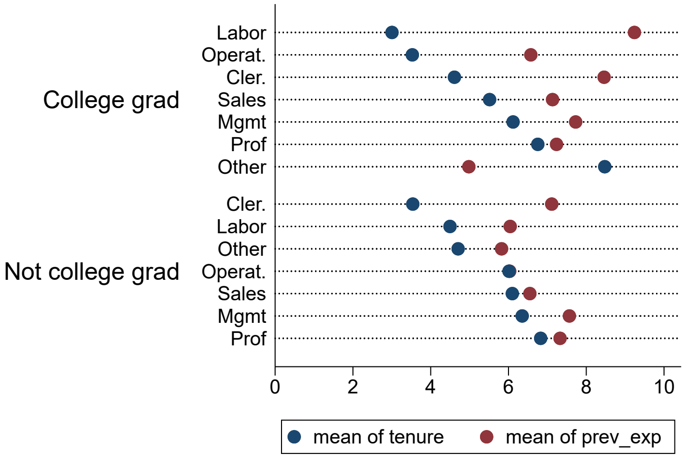
```
graph dot tenure prev_exp, over(occ7, sort(1)) over(collgrad)
```

We can use the `sort()` option when there are more `over()` variables. Here the markers are ordered according to the mean of `tenure` within each level of `collgrad`. *Uses nlsw.dta*



```
graph dot tenure prev_exp, over(occ7, sort(1)) over(collgrad,
descending)
```

We add the `descending` option to the second `over()` option, and the levels of `collgrad` are now shown with college graduates appearing first. *Uses nlsw.dta*

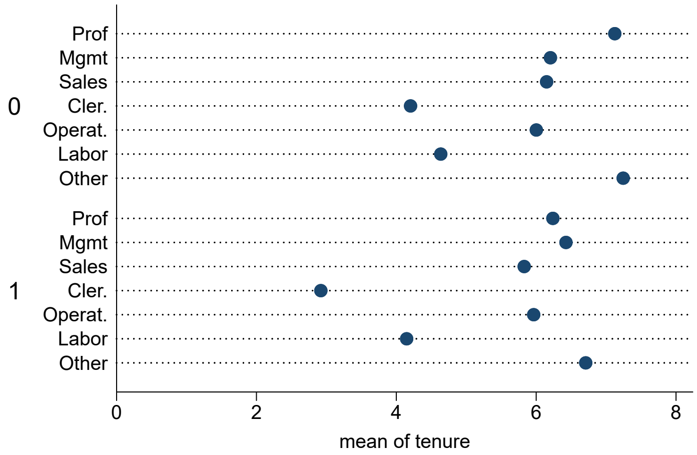


6.4 Controlling the categorical axis

This section describes ways that we can label the categorical axis in dot plots. Dot plots, like bar and box plots, are different from other plots because their x axis is formed by categorical variables. (Stata calls the axis with the categorical variable(s) the x axis, even though it may be placed on the left axis.) This section describes options we can use to customize the categorical axis. For more details on these options, see `help cat_axis_label_options` and `help cat_axis_line_options`. We will start by seeing how we can change the labels used for the categorical axis.

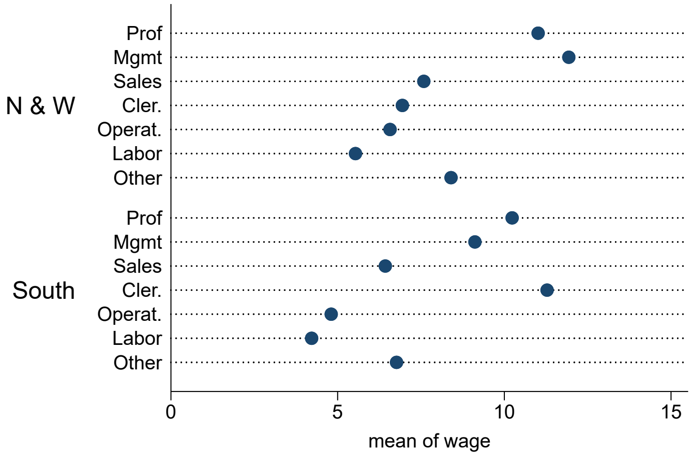
```
graph dot tenure, over(occ7) over(south)
```

This is an example of a dot plot with two `over()` variables graphing the average tenure broken down by occupation and whether one lives in the South. The variable `south` is a dummy variable that does not have any value labels, so the x axis is not labeled well. *Uses nlsw.dta*

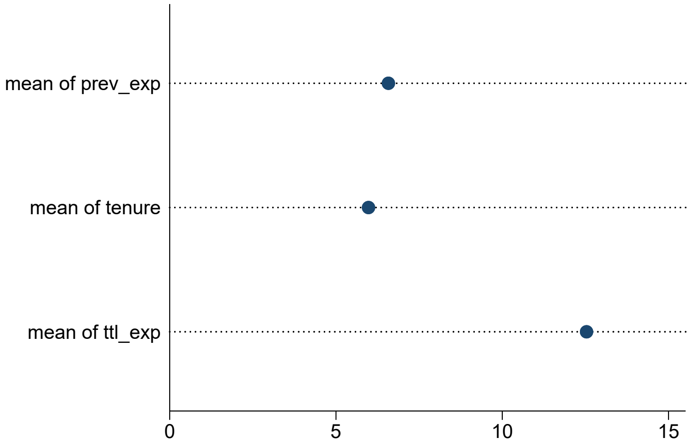


```
graph dot wage, over(occ7) over(south, relabel(1 "N & W" 2
"South"))
```

We can use the `relabel()` option to change the labels displayed for the levels of `south`, giving the x axis more meaningful labels. We wrote `relabel(1 "N & W")`, not `relabel(0 "N & W")`, because these numbers do not represent the actual levels of `south` but the ordinal position of the levels, that is, first and second. *Uses nlsw.dta*



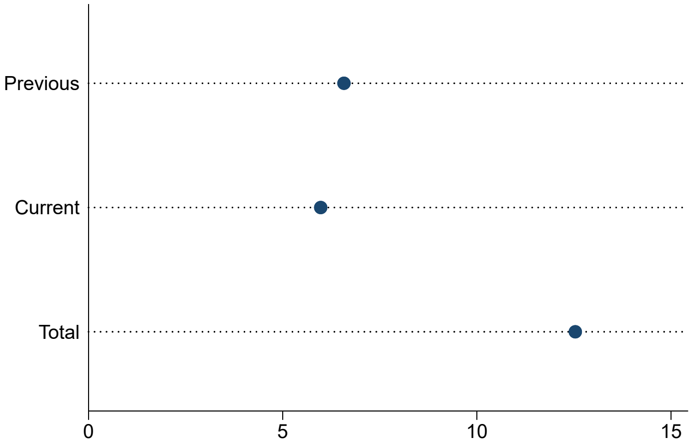
```
graph dot prev_exp tenure ttl_exp, ascategory
```



This `graph dot` command has multiple y variables but uses the `ascategory` option to plot the different y variables as if they were categorical variables. The dots for the different y variables are plotted on different lines using the same symbol, and each line is labeled on the x axis rather than using a legend. The default labels on the x axis are not bad, but we might want to change them. *Uses nlsw.dta*

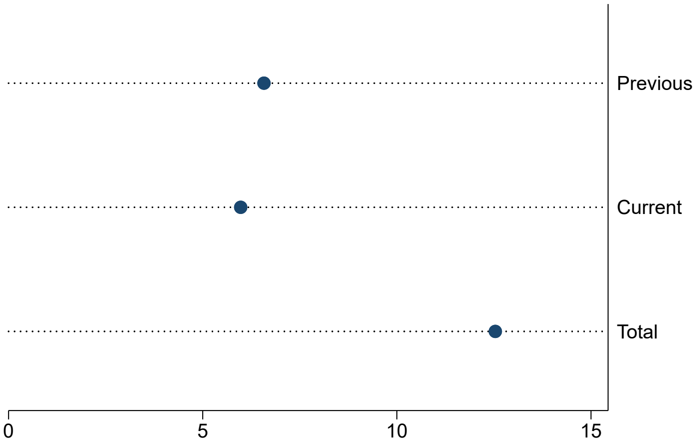
```
graph dot prev_exp tenure ttl_exp, ascategory
yvaroptions(relabel(1 "Previous" 2 "Current" 3 "Total"))
```

If we had an `over()` option, we could use the `relabel()` option to change the labels on the x axis. However, because we have multiple y variables that we have treated as categories, we use the `yvaroptions(relabel())` option to modify the labels on the x axis. *Uses nlsw.dta*



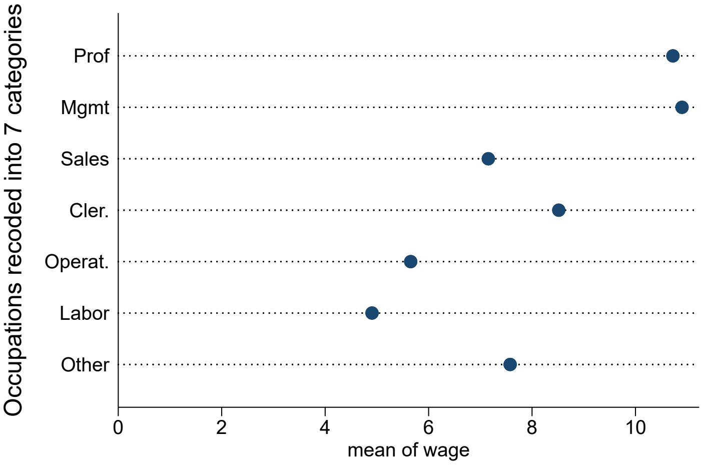
```
graph dot prev_exp tenure ttl_exp, ascategory xalternate
yvaroptions(relabel(1 "Previous" 2 "Current" 3 "Total"))
```

We add the `xalternate` option, which moves the labels for the x axis to the opposite side, here, from the left to the right. *Uses nlsw.dta*



```
graph dot wage, over(occ7) l1title("Occupations recoded into
7 categories")
```

In this example, the categorical axis represents the occupation after recoding it into 7 categories. We can use the `l1title()` option to add a title to the left of the graph labeling this axis; see Standard options : Titles (section 9.1) for more details. *Uses nlsw.dta*

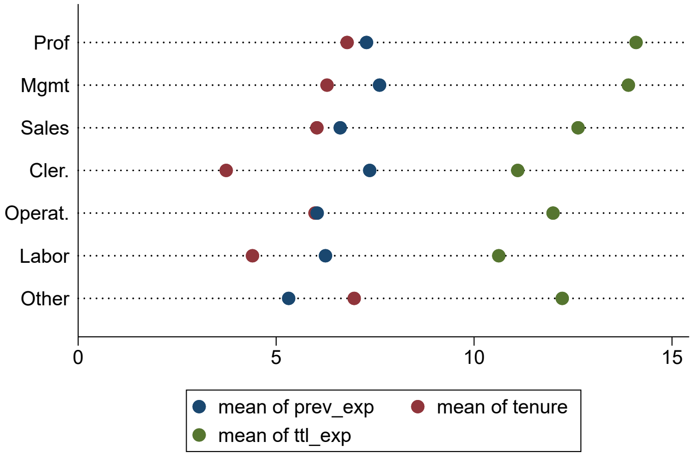

Occupations recoded into 7 categories
Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other
0
2
4
6
8
10
mean of wage

6.5 Controlling the legend

This section discusses the use of the legend for dot plots, emphasizing the features that are unique to dot plots. The section Options : Legend (section 8.9) goes into great detail about the legend, as does `help legend_options`. We can use the legend for multiple y variables or when the first `over()` variable is treated as a y variable via the `asyvars` option. See Dot : Yvars and over (section 6.1) for more information about using multiple y variables and more examples of treating the first `over()` variable as a y variable.

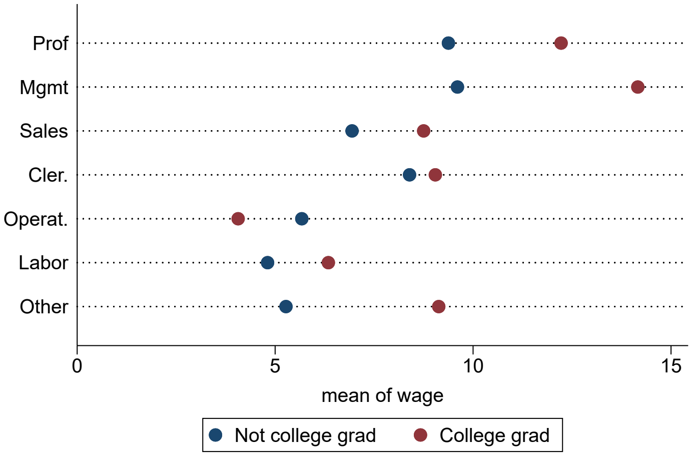
```
graph dot prev_exp tenure ttl_exp, over(occ7)
```

Consider this dot plot of three different variables. These variables are shown with different markers, and a legend is used to identify the variables. *Uses nlsw.dta*



```
graph dot wage, over(collgrad) over(occ7) asyvars
```

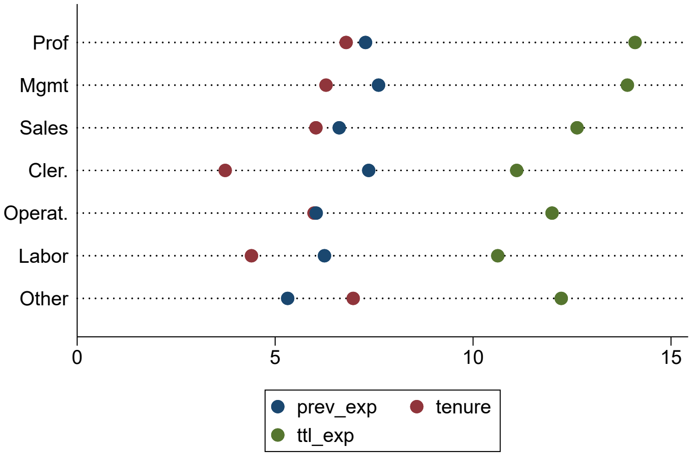
This is another example of how a legend can arise in a Stata dot plot if the `over()` variable is used with the `asyvars` option. Stata treats the levels of the first `over()` variable as if they were really multiple y variables. *Uses nlsw.dta*



Unless otherwise mentioned, the legend options described below work the same whether we derived the legend from multiple *y* variables or from an `over()` variable that was combined with the `asyvars` option.

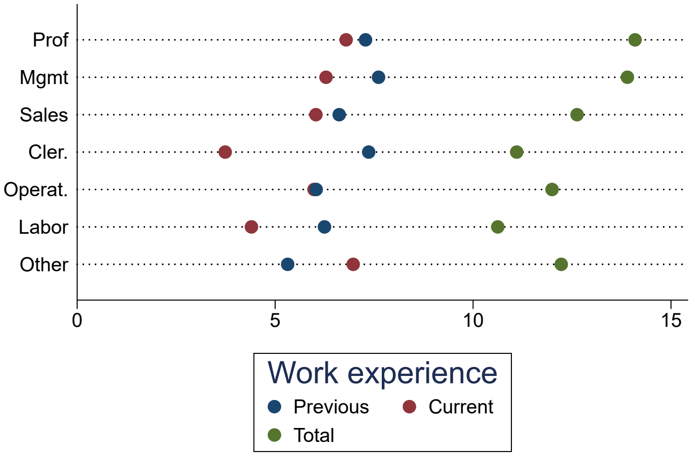
```
graph dot prev_exp tenure ttl_exp, over(occ7) nolabel
```

The `nolabel` option works only when we have multiple *y* variables. When we use this option, the variable names (not the derived variable labels) are used in the legend. For example, instead of showing the derived variable label *mean of prev_exp*, this option shows the variable name *prev_exp*. *Uses nlsw.dta*



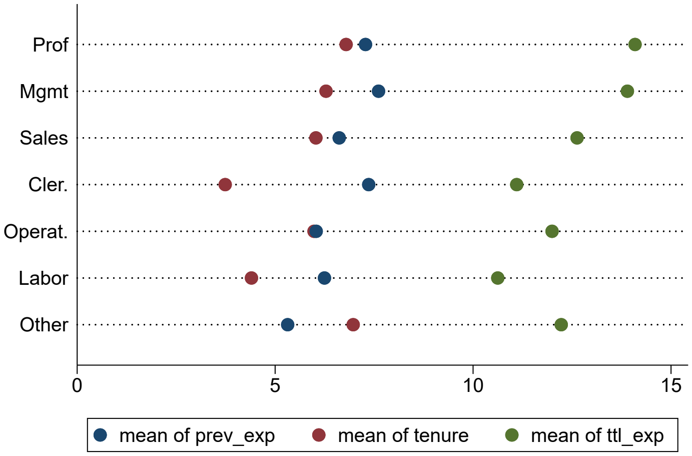
```
graph dot prev_exp tenure ttl_exp, over(occ7) legend(label(1
"Previous") label(2 "Current") label(3 "Total") title("Work
experience"))
```

We use the `legend(label())` option to change the labels for the variables in the legend and the `title()` option to add a title to the legend. *Uses nlsw.dta*



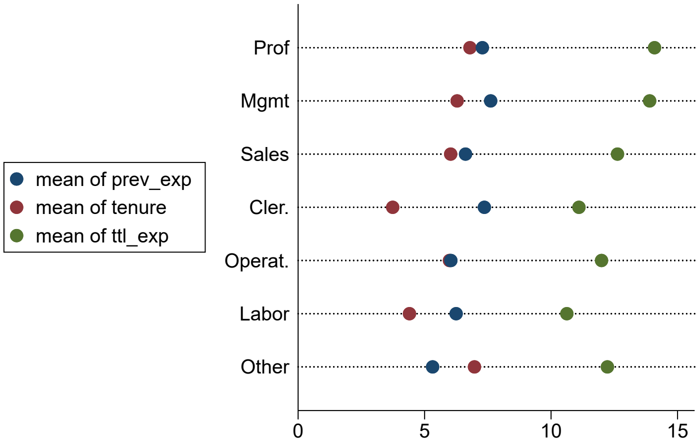
```
graph dot prev_exp tenure ttl_exp, over(occ7) legend(rows(1))
```

Using the `legend(rows(1))` option makes the legend display as one row. *Uses nlsw.dta*



```
graph dot prev_exp tenure ttl_exp, over(occ7) legend(cols(1)
position(9))
```

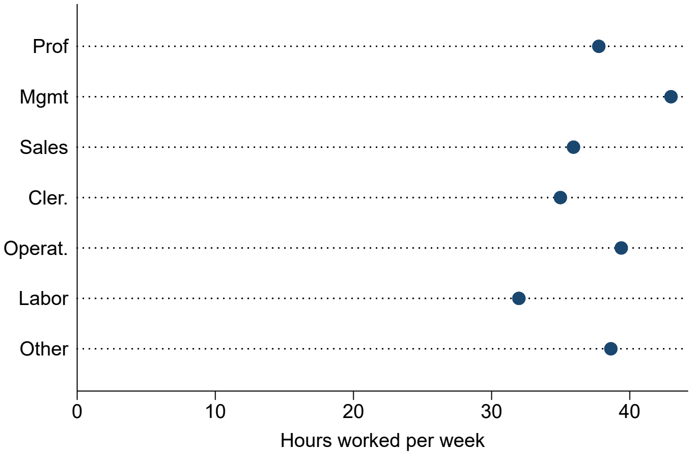
Here the legend is one column and moved to the left by using the `legend(cols(1) position(9))` option. *Uses nlsw.dta*


mean of prev_exp
mean of tenure
mean of ttl_exp
Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other
0
5
10
15

6.6 Controlling the y axis

This section describes options to customize the y axis with dot plots. To be precise, when Stata refers to the y axis on a dot plot, it refers to the axis with the continuous variable, which is placed on the bottom (where the x axis would traditionally be placed). This section emphasizes the features that are particularly relevant to dot plots. For more details, see Options : Axis titles (section 8.4), Options : Axis labels (section 8.5), and Options : Axis scales (section 8.6). Also see `help axis_title_options`, `help axis_label_options`, and `help axis_scale_options`.

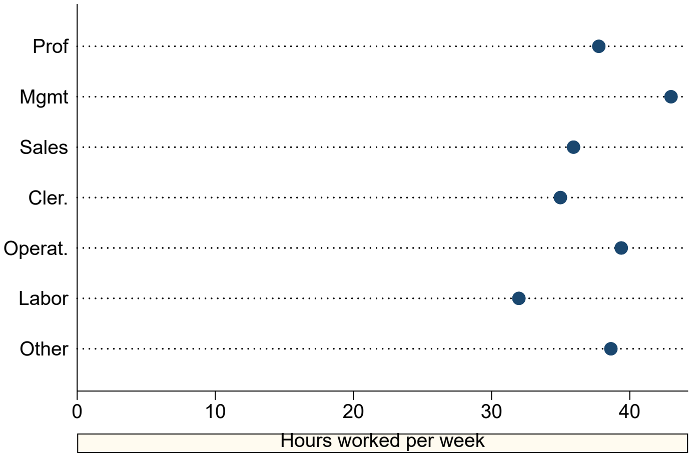
```
graph dot hours, over(occ7) ytitle(Hours worked per week)
```



Consider this graph showing the mean hourly wage broken down by occupation. We use the `ytitle()` option to add a title to the y axis. See Options : Axis titles (section 8.4) and `help axis_title_options` for more details. *Uses nlsw.dta*

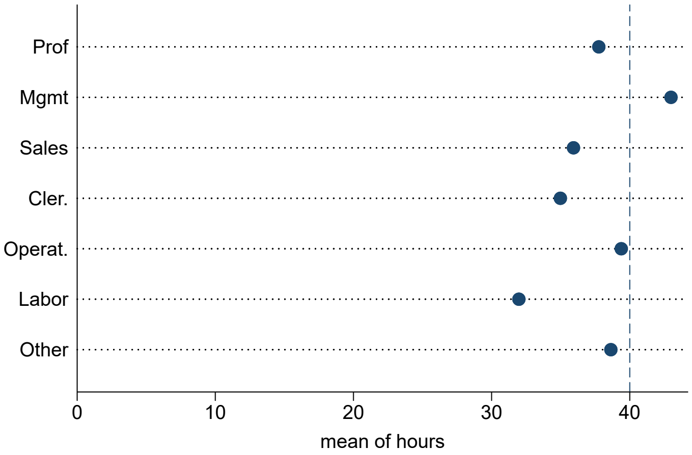
```
graph dot hours, over(occ7) ytitle(Hours worked per week,
fcolor(eggshell) box bexpand)
```

Because the title is considered a textbox, you can use textbox options as illustrated here to control the look of the title. See Options : Textboxes (section 8.11) for more examples of how to use textbox options to control the display of text. *Uses nlsw.dta*



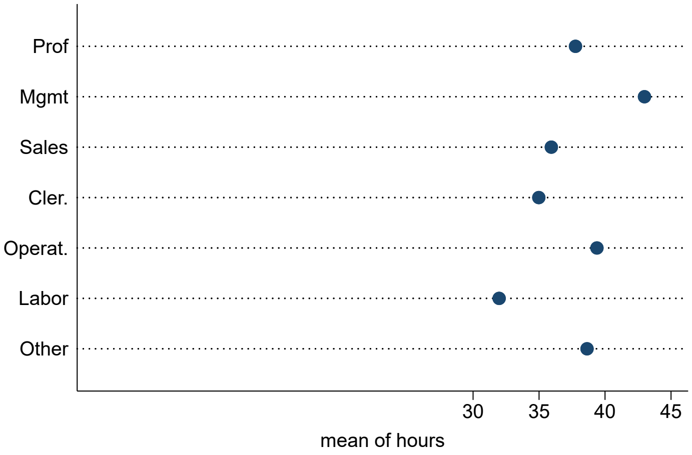
```
graph dot hours, over(occ7) yline(40, lwidth(thin)
lcolor(navy) lpattern(dash))
```

This example uses the `yline()` option to add a thin, navy, dashed line to the graph where the hours worked equals 40. *Uses nlsw.dta*



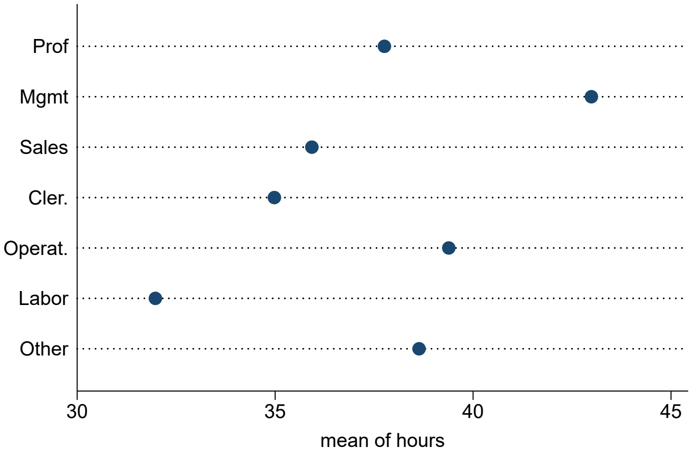
```
graph dot hours, over(occ7) ylabel(30(5)45)
```

We use the `ylabel()` option to label the y axis from 30 to 45, incrementing by 5. See Options : Axis labels (section 8.5) and `help axis_label_options` for more details. The y axis still begins at 0, but see the next example for how we can override this. *Uses nlsw.dta*



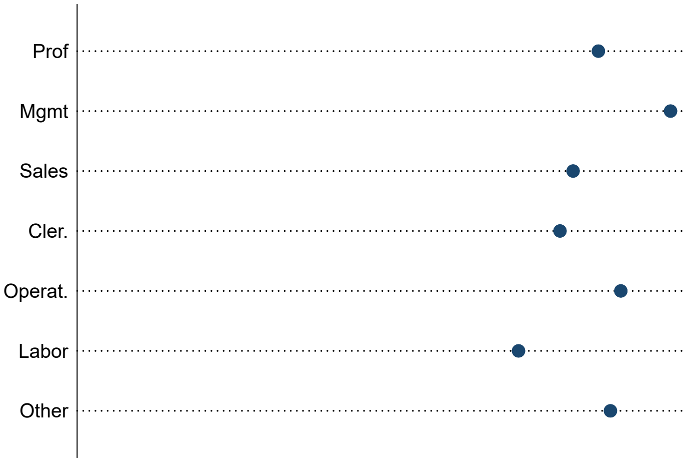
```
graph dot hours, over(occ7) ylabel(30(5)45) exclude0
```

When we add the `exclude0` option, the dot plot does not automatically begin at 0. Here it starts at 30 because that is the value we specified as the starting point on the `ylabel()` option. *Uses nlsw.dta*



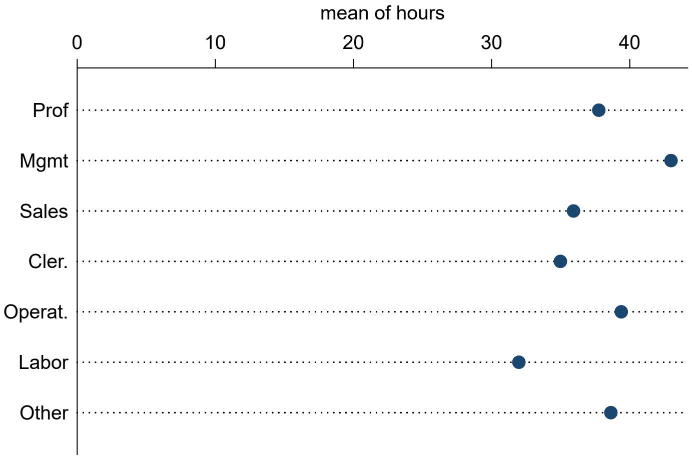
```
graph dot hours, over(occ7) yscale(off)
```

We can use the `yscale(off)` option to turn off the y axis. See **Options : Axis scales** (section 8.6) and `help axis_scale_options` for more details. Disregard any references to `xscale()` because this option is not valid when using `graph dot`. *Uses nlsw.dta*



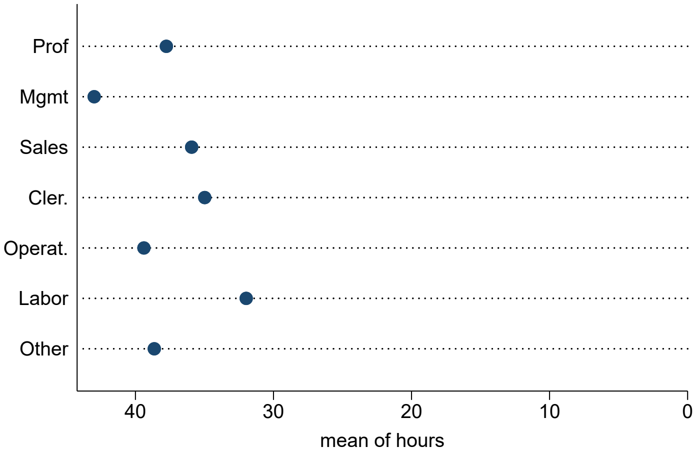
```
graph dot hours, over(occ7) yalternate
```

The `yalternate` option puts the y axis on the opposite side, here, on the top of the graph. *Uses nlsw.dta*



```
graph dot hours, over(occ7) yreverse
```

We can reverse the direction of the y axis with the `yreverse` option. *Uses nlsw.dta*

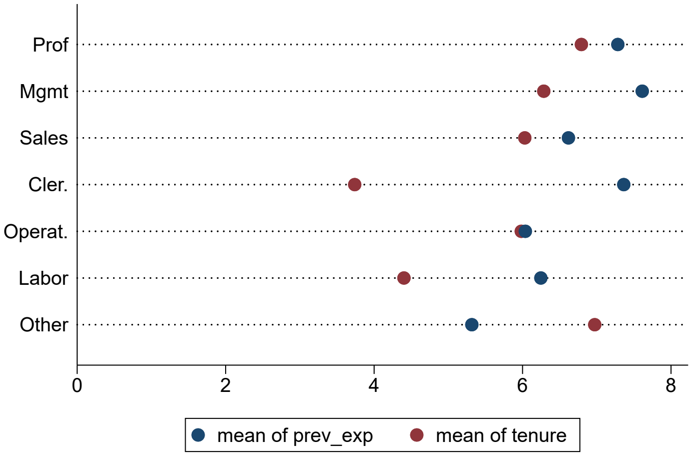

Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other
40
30
20
10
0
mean of hours

6.7 Changing the look of dot rulers

This section shows how we can control the look of the lines in our dot plots. We will see how we can control the space between the lines, the color of the lines, and other characteristics of the line. For more information, see the *linelook_options* table in `help graph dot`.

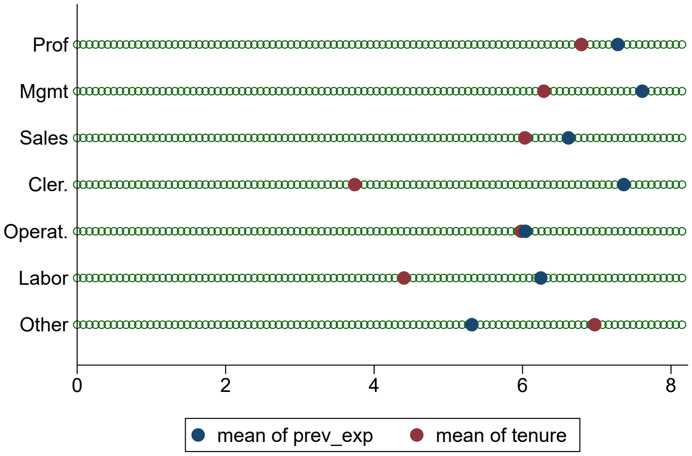
```
graph dot prev_exp tenure, over(occ7)
```

Consider this dot plot showing previous and current work experience broken down by occupation. Each dot plot has a series of small dots that forms a line on which the symbols are plotted. *Uses nlsw.dta*

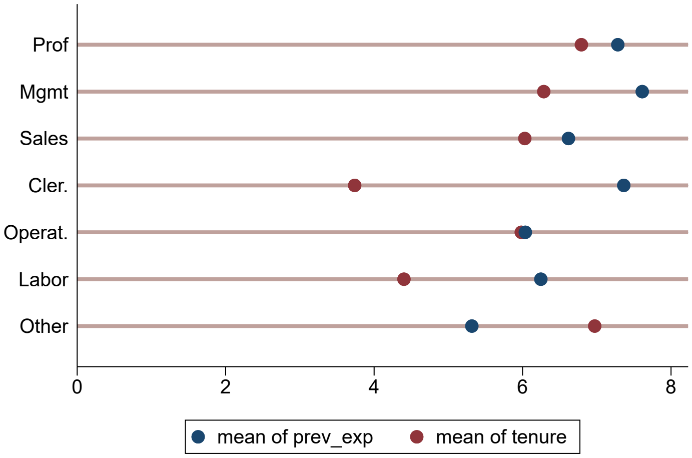


```
graph dot prev_exp tenure, over(occ7) dots(msymbol(Oh)
msize(medium) mcolor(dkgreen))
```

With the `dots()` option, the small dots are displayed as medium-sized, dark green, hollow circles. See **Styles : Symbols** (section [10.11](#)), **Styles : Markersize** (section [10.9](#)), and **Styles : Colors** (section [10.2](#)) for more information. *Uses nlsw.dta*



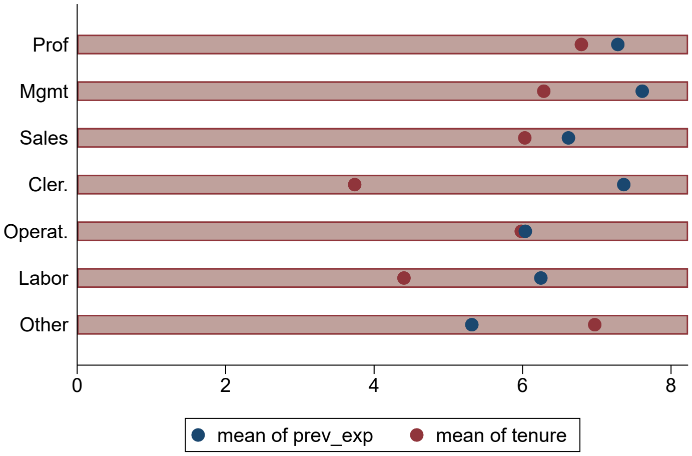
```
graph dot prev_exp tenure, over(occ7) linetype(line)
lines(lwidth(thick) lcolor(erose))
```



The `linetype(line)` option displays lines instead of dots, and the `lines()` option makes the line width thick and the line color rose. You could also add the `lpattern()` option to control the line pattern. See **Styles : Linewidth** (section 10.7), **Styles : Colors** (section 10.2), and **Styles : Linepatterns** (section 10.6) for more information. *Uses nlsw.dta*

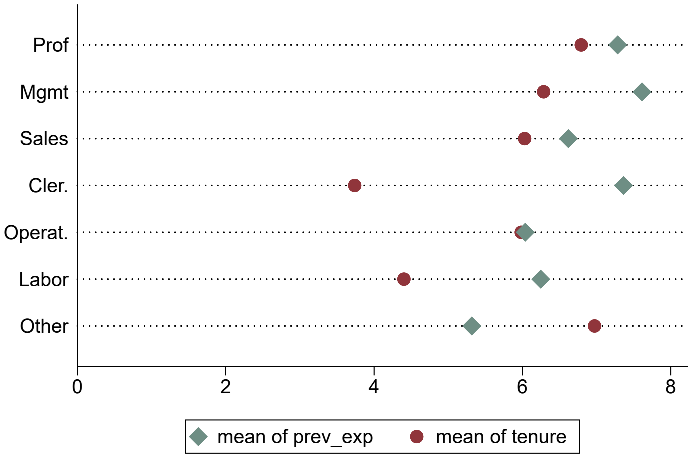
```
graph dot prev_exp tenure, over(occ7) linetype(rectangle)
rwidth(3) rectangles(fcolor(erose) lcolor(maroon))
```

Here we change the `linetype()` to a rectangle. The `rwidth(3)` option sets the rectangle width to three times its normal width. We also use the `rectangle()` option to customize the rectangle, using the `fcolor()` (fill color) and `lcolor()` (line color) options to make the rectangle rose on the inside with a maroon outline. *Uses nlsw.dta*



Now I show options that allow you to control the markers and whether the markers are displayed on the same line.

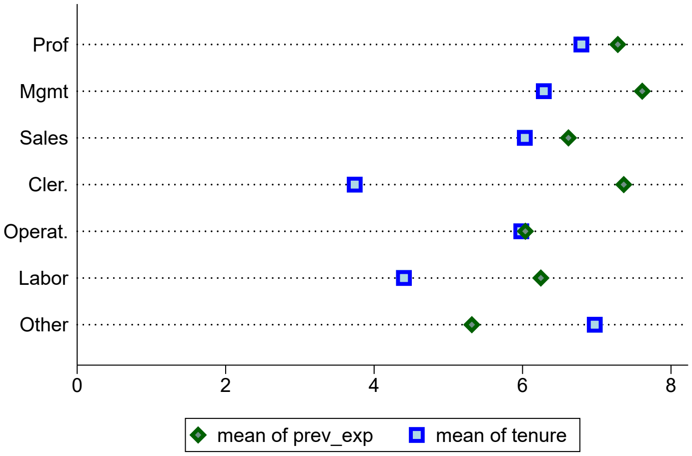
```
graph dot prev_exp tenure, over(occ7) marker(1, msymbol(D)
mcolor(teal) msize(large))
```



Here we use the `marker()` option to control the marker used for the first *y* variable, making it a large teal-colored diamond. See Options : Markers (section 8.1) for more details on how we can control markers. *Uses nlsw.dta*

```
graph dot prev_exp tenure, over(occ7) marker(1, msymbol(d)
mfcolor(teal) mlcolor(dkgreen) mlwidth(thick)) marker(2,
msymbol(S) mfcolor(ltblue) mlcolor(blue) mlwidth(thick))
```

In this example, we use two `marker()` options, so we can control both markers. The first marker is now a diamond with a teal fill and a thick, dark green outline. The second marker is a square, light blue on the inside with a thick blue outline. The section Options : Markers (section [8.1](#)) has more details on controlling markers. *Uses nlsw.dta*

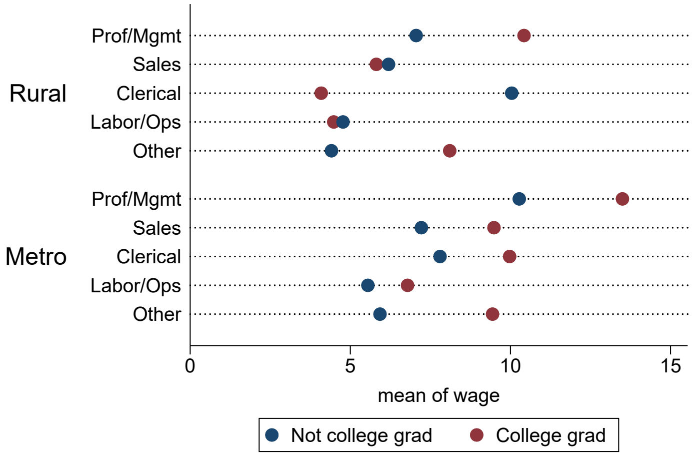


6.8 Graphing by groups

This section discusses the use of the `by()` option in combination with `graph dot`. Normally, you would use the `over()` option instead of the `by()` option, but sometimes the `by()` option is either necessary or more advantageous. For example, a `by()` option is useful if you exceed the maximum number of `over()` options (three if you have one y variable or two if you have multiple y variables). Thus the `by()` option allows you to break down your data by more categorical variables. `by()` also provides more flexibility in the placement of the separate panels. For more information about the `by()` option, see Options : By (section 8.8), and for more information about the `over()` option, see Dot : Yvars and over (section 6.1).

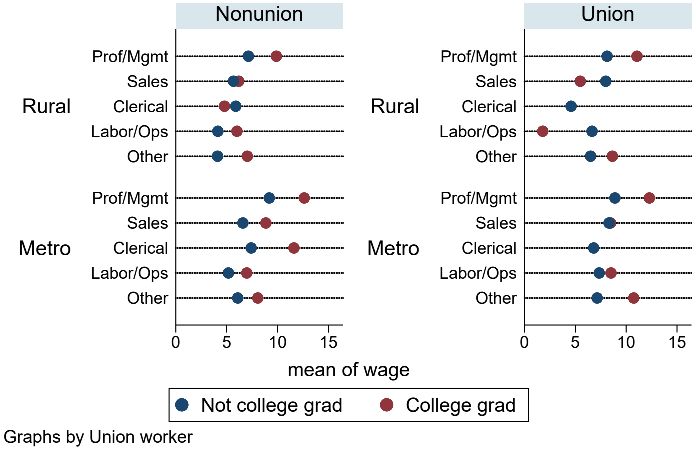
```
graph dot wage, over(collgrad) over(occ5) over(urban2)
```

Consider this dot graph breaking down wages by three categorical variables. If we wanted to break this down by another categorical variable, we could not use another `over()` option because we can have only a maximum of three `over()` options with one y variable. *Uses nlsw.dta*



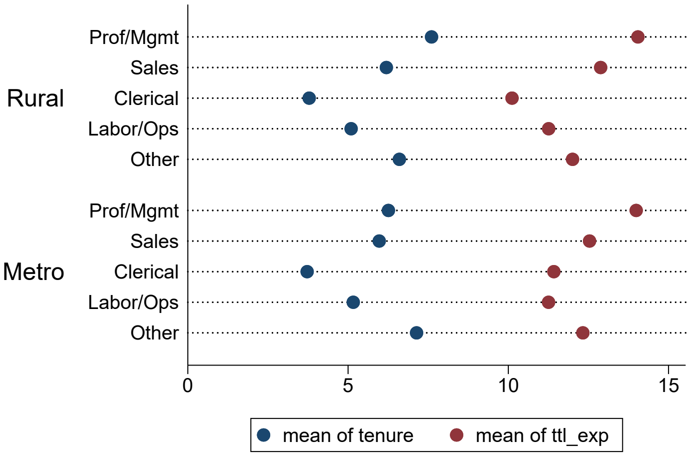
```
graph dot wage, over(collgrad) over(occ5) over(urban2)
by(union)
```

To break down `wage` further by `union`, we can use the `by(union)` option. *Uses nlsw.dta*



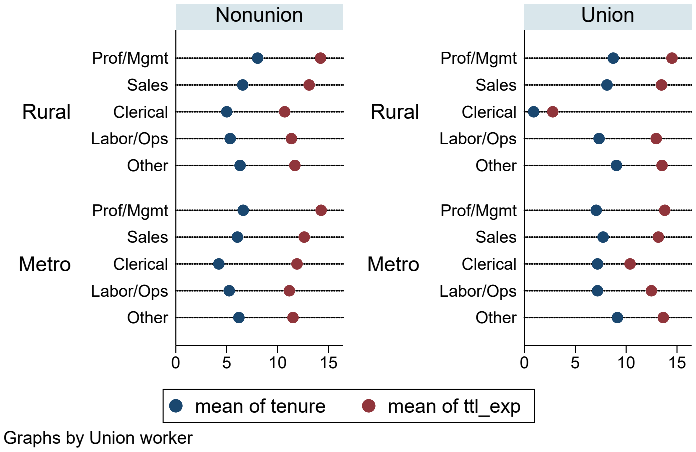
```
graph dot tenure ttl_exp, over(occ5) over(urban2)
```

Consider this dot graph with multiple *y* variables, breaking down them by two categorical variables using two `over()` options. When we have multiple *y* variables, we can have a maximum of two `over()` options. *Uses nlsw.dta*



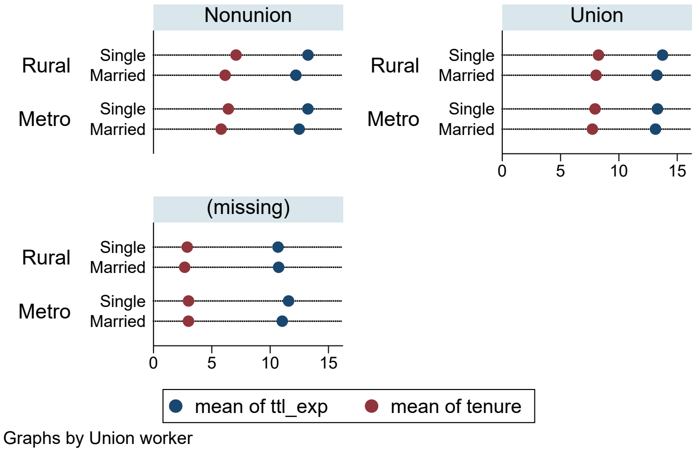
```
graph dot tenure ttl_exp, over(occ5) over(urban2) by(union)
```

To further break down these results by union membership, we can add the `by(union)` option. Although this example shows only one variable in the `by()` option, we can specify multiple variables. *Uses nlsw.dta*



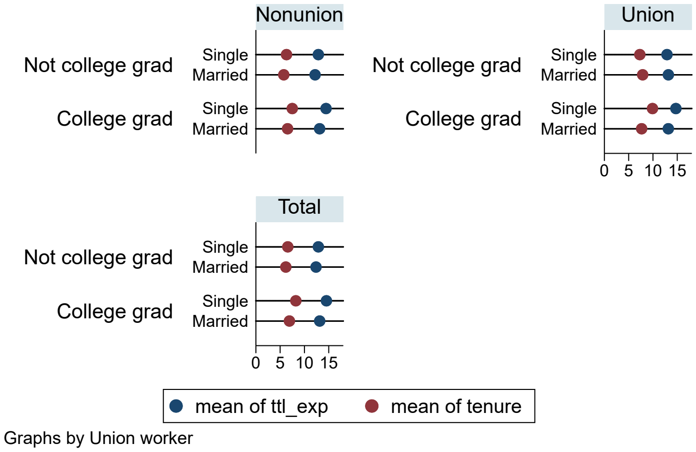
```
graph dot ttl_exp tenure, over(married) over(urban2)
by(union, missing)
```

We can use the `missing` option to include a panel for the missing values of `union`. Note that we changed the first `over()` variable to be `over(married)` to make an example that was more readable. *Uses nlsw.dta*



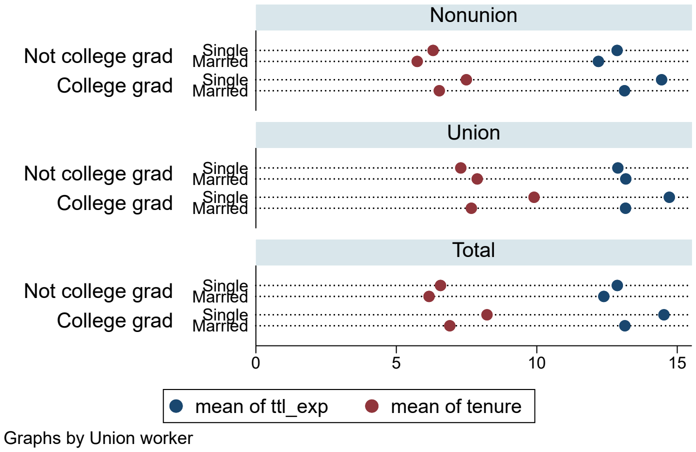
```
graph dot ttl_exp tenure, over(married) over(collgrad)
by(union, total)
```

We can add the `total` option to include a panel for all observations. *Uses nlsw.dta*



```
graph dot ttl_exp tenure, over(married) over(collgrad)
by(union, total cols(1))
```

We can use the `cols(1)` option to show the graphs in one column. *Uses nlsw.dta*



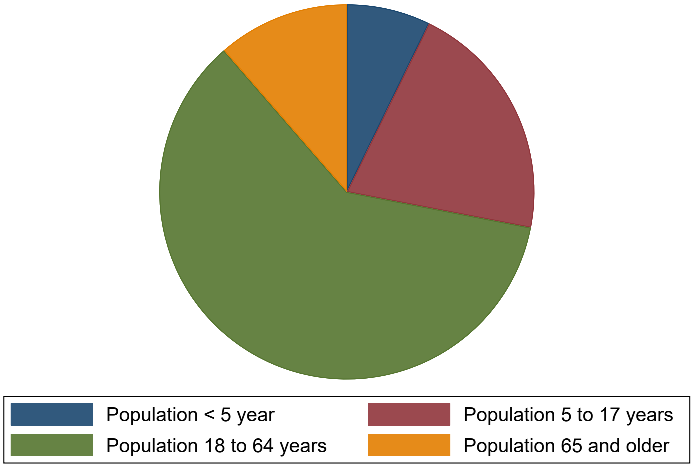
Chapter 7
Pie charts

This chapter discusses the use of pie charts in Stata. The first section illustrates the different ways we can create pie charts in Stata, followed by a section showing how we can sort the slices in our pie charts. Next the chapter shows how we can customize the display of individual slices, as well as control the colors of the pie chart. The next sections demonstrate different ways we can label the pie slices and how we can control the legend for pie charts. Finally, the chapter discusses how to use the `by()` option.

7.1 Types of pie charts

This section describes different ways to produce pie charts with Stata. Stata allows you to produce pie charts of multiple y variables, with each y variable corresponding to a slice. We can also create a pie chart based on one y variable broken down by one `over()` variable. Finally, we can create a pie chart with no y variables broken down by an `over()` variable, which counts the number of observations by each level of the `over()` variable. For more details, see `help graph pie`.

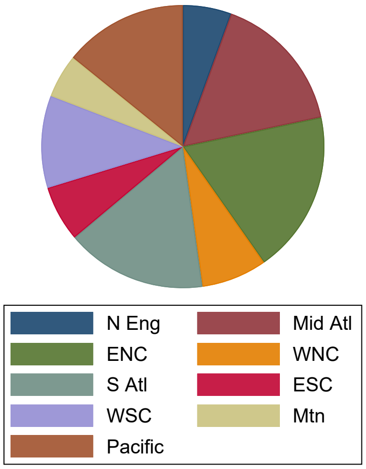
```
graph pie poplt5 pop5_17 pop18_64 pop65p
```



Here we supply multiple y variables, and each y variable corresponds to a slice in the pie. The first y variable is the population in the state that is younger than 5 years old, the next y variable is the population 5–17 years old, the next y variable is 18–64 years old, and the last y variable is 65 years and older. The entire pie would correspond to the sum of all these variables across all states. The first slice then corresponds to the percentage of the total population that is younger than 5 years old. *Uses allstates.dta*

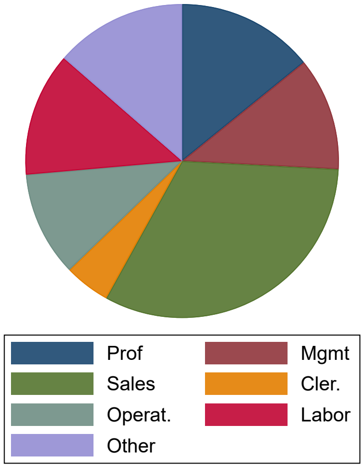
```
graph pie pop, over(division)
```

In this example, we supply a y variable and an `over()` option. The y variable corresponds to the population of the state, the entire pie corresponds to the entire population, and each slice corresponds to the percentage of the population for each level of `division`. *Uses allstates.dta*



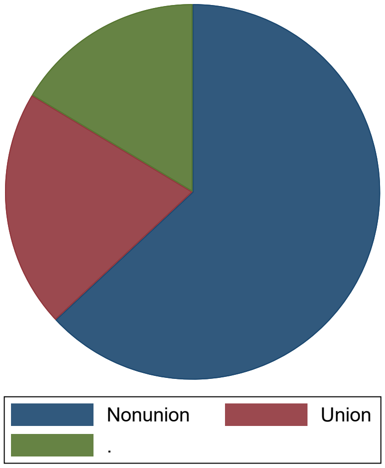
```
graph pie, over(occ7)
```

We now switch to the `nlsw` data file. Here we use an `over()` option but no y variable (in a sense, the observation itself serves as the y variable). This pie chart is much like a visual frequency distribution of `occ7`, where the size of each slice corresponds to the proportion of women in each occupation. *Uses nlsw.dta*



```
graph pie, over(union) missing
```

This example shows the proportion of women in union and nonunion jobs. With the `missing` option, another pie slice is added for the observations in which `union` is missing. *Uses nlsw.dta*

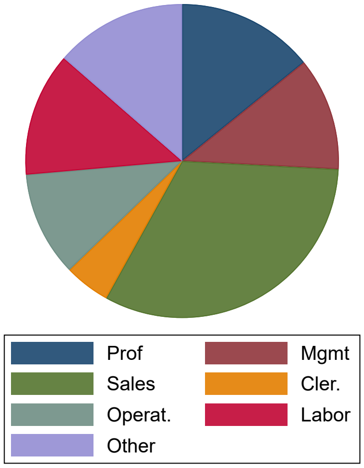

Nonunion
Union
.

7.2 Sorting pie slices

This section describes how we can sort and arrange slices in pie charts. For more details, see `help graph pie`. Most of the examples in this section use the `vg_lgndc` scheme, which places the legend at the left in one column.

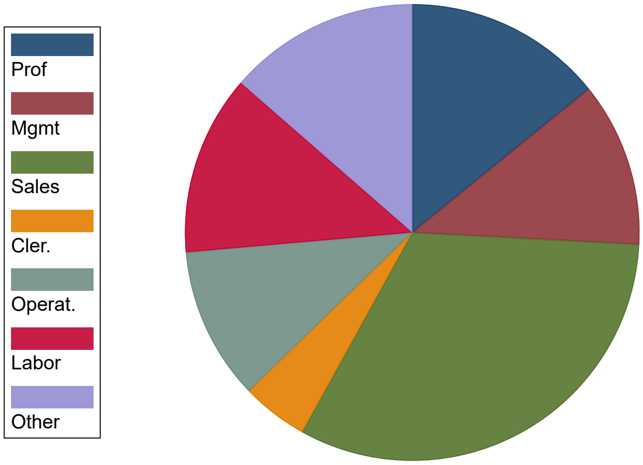
```
graph pie, over(occ7)
```

Consider this pie chart showing the number of women who work in seven different occupations. The slices are ordered according to the levels of `occ7` from 1 to 7, rotating clockwise, starting with the first slice, which is positioned at 90 degrees. *Uses nlsw.dta*



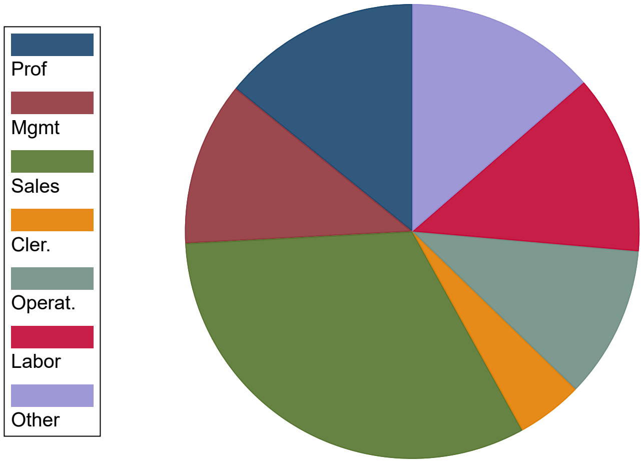
```
graph pie, over(occ7) scheme(vg_lgndc)
```

By adding the `scheme(vg_lgndc)` option, the legend is positioned to the left. *Uses nlsw.dta*



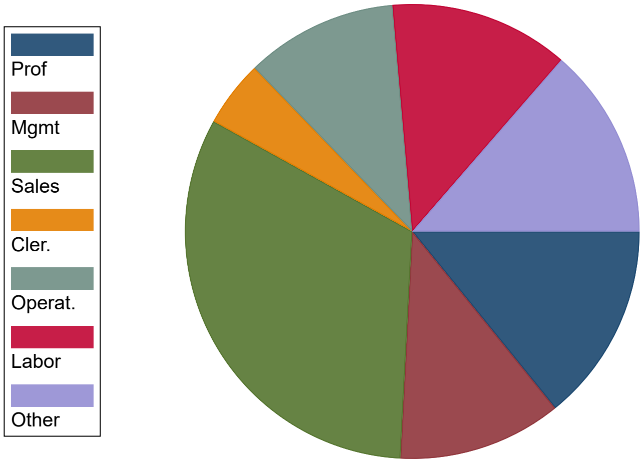
```
graph pie, over(occ7) scheme(vg_lgndc) noclockwise
```

With the `noclockwise` option, we can display the slices in counterclockwise order. *Uses nlsw.dta*



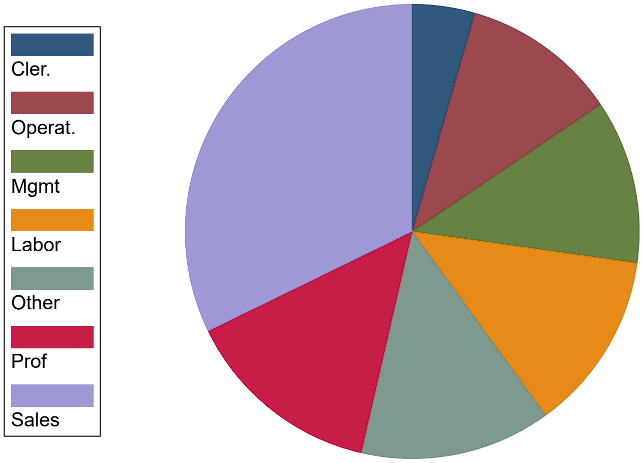
```
graph pie, over(occ7) scheme(vg_lgndc) angle0(0)
```

The `angle0()` option sets the angle of the line that begins the first pie slice. Here we make the first pie slice begin at 0 degrees. *Uses nlsw.dta*



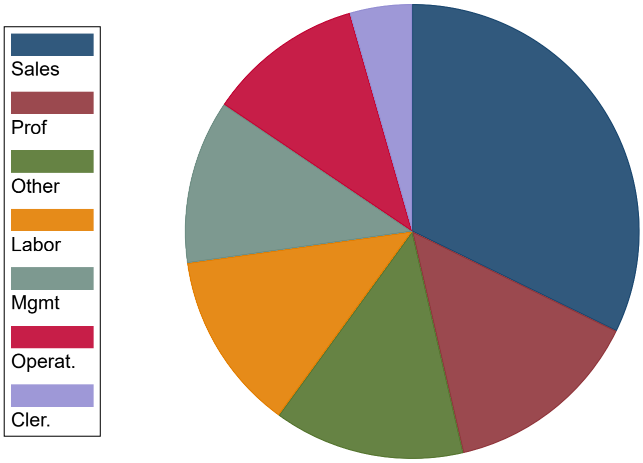
```
graph pie, over(occ7) scheme(vg_lgndc) sort
```

The `sort` option sorts the slices according to their size, from smallest to largest. *Uses nlsw.dta*



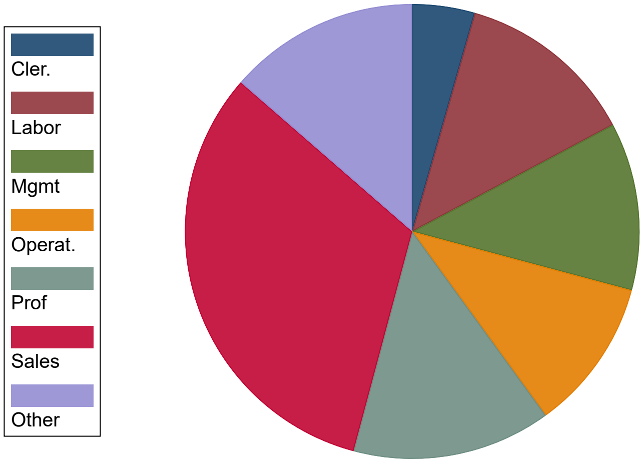
```
graph pie, over(occ7) scheme(vg_lgndc) sort descending
```

Adding the `descending` option to the `sort` option orders the slices from largest to smallest. *Uses nlsw.dta*



```
graph pie, over(occ7) scheme(vg_lgndc) sort(occ7alpha)
```

Say that we wanted to sort the slices (alphabetically) by occupation name. We have created a new variable, `occ7alpha`, which is a recoded version of `occ7`. It is recoded such that, as `occ7alpha` goes from 1 to 7, the occupations are alphabetized (except for *Other*, which is placed last). We add `sort(occ7alpha)`, and the slices are ordered alphabetically. *Uses nlsw.dta*

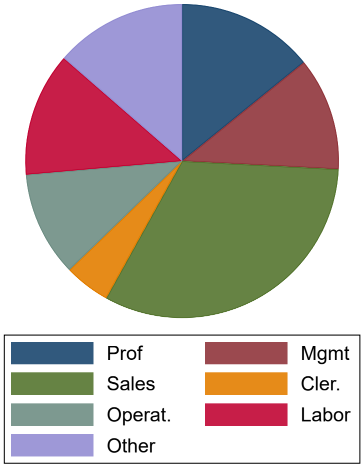

Cler.
Labor
Mgmt
Operat.
Prof
Sales
Other

7.3 Changing the look and color and exploding pie slices

This section describes how to change the color of pie slices, explode pie slices, and control the characteristics of lines surrounding the pie slices. For more details, see `help graph pie`.

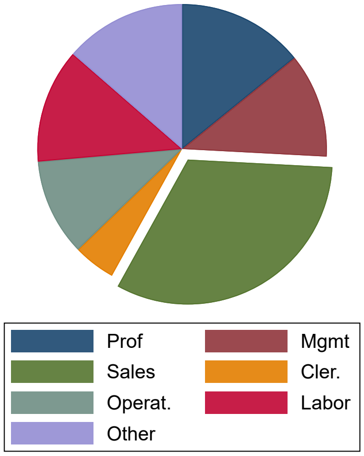
```
graph pie, over(occ7)
```

Consider this pie chart showing the number of women who work in these seven different occupations. The slices are colored using the colors indicated by the scheme. None of the slices are exploded, and no lines surround the slices. *Uses nlsw.dta*



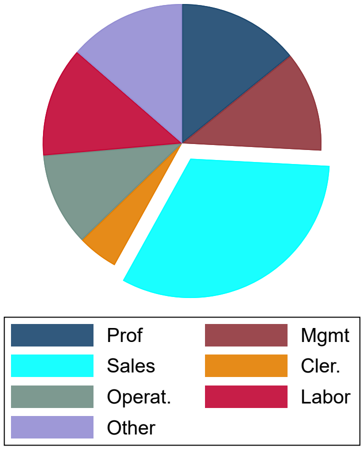
```
graph pie, over(occ7) pie(3, explode)
```

In this example, we use the `pie()` option to explode the third pie slice, calling attention to this slice. *Uses nlsw.dta*



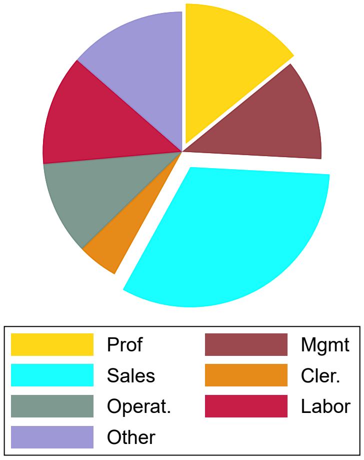
```
graph pie, over(occ7) pie(3, explode(vlarge) color(cyan))
```

Here we specify `explode(vlarge)` to increase the distance this slice is exploded. We also make the third slice cyan to make it more noticeable. See Styles : Colors (section 10.2) for other available colors. *Uses nlsw.dta*



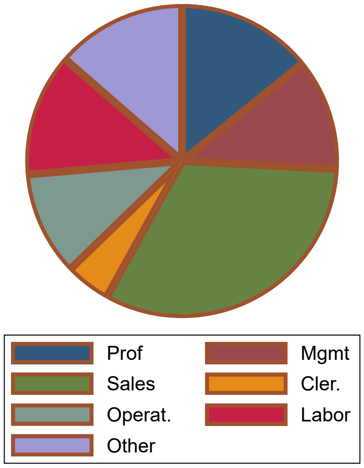
```
graph pie, over(occ7) pie(3, explode(vlarge) color(cyan))
pie(1, explode(small) color(gold))
```

We can use the `pie()` option repeatedly. Here we change the color and explode slices 1 and 3. *Uses nlsw.dta*



```
graph pie, over(occ7) line(lcolor(sienna) lwidth(thick))
```

We can use the `line()` option to change the characteristics of the lines surrounding the pie slices. Here we add the `lcolor()` (line color) and `lwidth()` (line width) options to make the line sienna and thick. *Uses nlsw.dta*

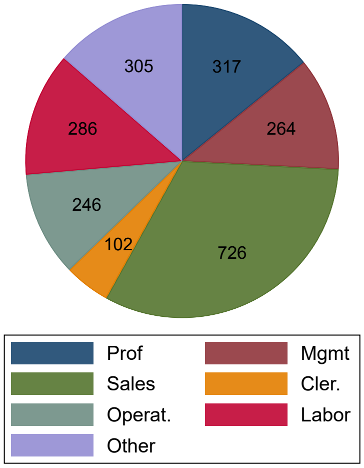

Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other

7.4 Slice labels

This section describes how we can label the pie slices. For more details, see `help graph pie`.

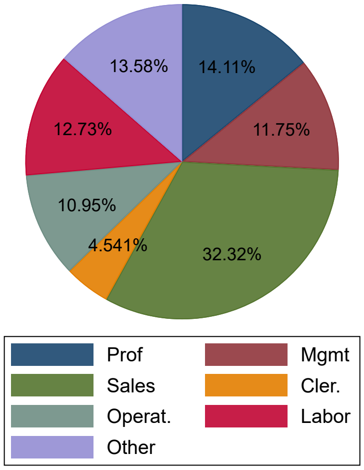
```
graph pie, over(occ7) plabel(_all sum)
```

Consider this pie chart showing the number of women who work in seven different occupations. Here we use the `plabel()` (pie label) option to label all slices with the sum, that is, the frequency of women who work in each occupation. *Uses nlsw.dta*



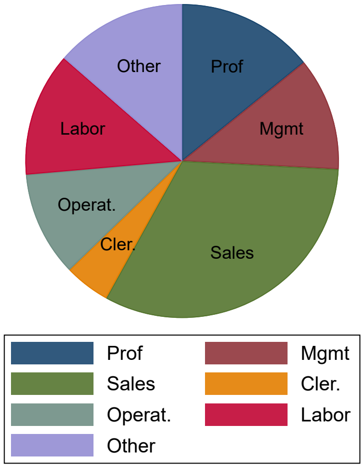
```
graph pie, over(occ7) plabel(_all percent)
```

By using the `percent` option, we can show the percentage of women who work in each occupation. *Uses nlsw.dta*



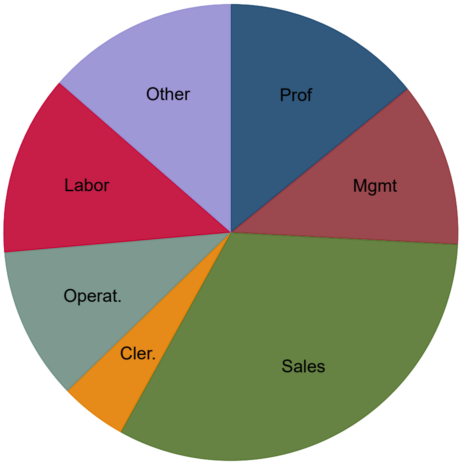
```
graph pie, over(occ7) plabel(_all name)
```

The `name` option adds a label, here, the name of the occupation. *Uses nlsw.dta*



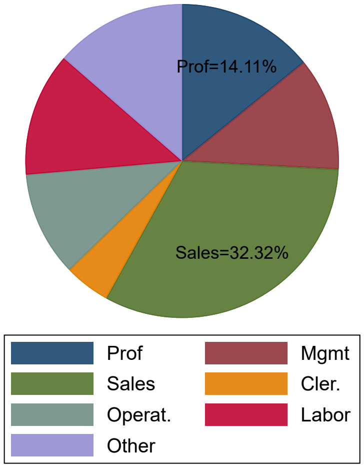
```
graph pie, over(occ7) plabel(_all name) legend(off)
```

When we use the `name` option, the legend is not necessary, and we can suppress it by using the `legend(off)` option. *Uses nlsw.dta*



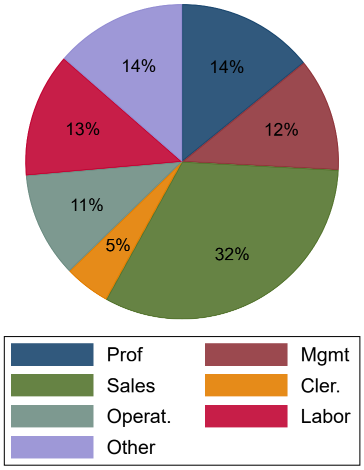
```
graph pie, over(occ7) plabel(1 "Prof=14.11%") plabel(3
"Sales=32.32%")
```

We can also use the `plabel()` option to put text into all slices or individual slices. Here we add text to the first and third slices. *Uses nlsw.dta*



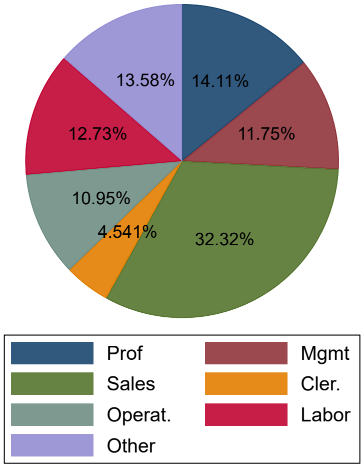
```
graph pie, over(occ7) plabel(_all percent, format("%2.0f"))
```

When we use `plabel()` to label slices with a `sum` or `percent`, we can use the `format()` option to control the format of the numeric values displayed. Here we display the percentages as whole numbers. *Uses nlsw.dta*



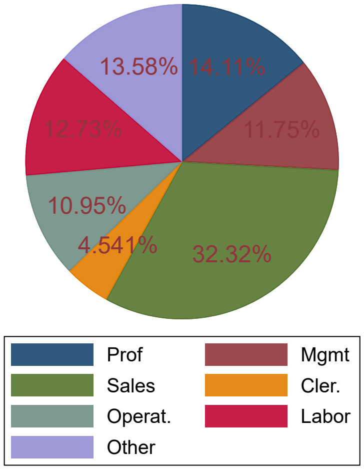
```
graph pie, over(occ7) plabel(_all percent, gap(-5))
```

We can use the `gap()` option to adjust the position of the label with respect to the center of the pie. A positive number pushes the label away from the center of the pie, and a negative value pushes the label closer to the center of the pie. *Uses nlsw.dta*



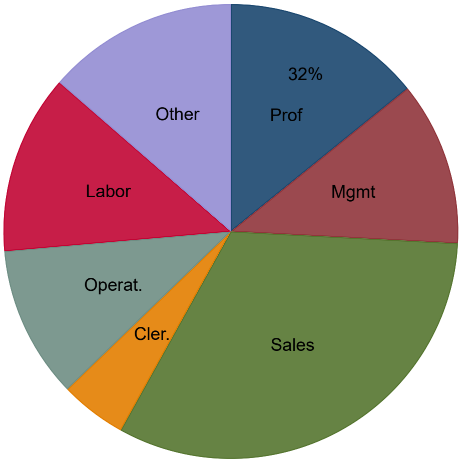
```
graph pie, over(occ7) plabel(_all percent, size(large)
color(maroon))
```

We can use textbox options to modify the display of the text labeling the pie slices. Here we increase the size of the text and change its color to maroon. See Options : Textboxes (section 8.11) for more options you can use. *Uses nlsw.dta*



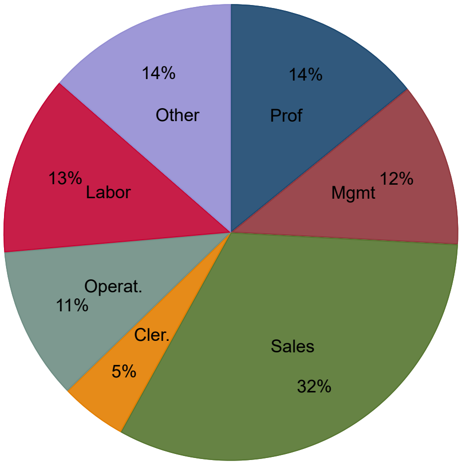
```
graph pie, over(occ7) plabel(_all name, gap(-5)) plabel(1
"32%", gap(5)) legend(off)
```

We can include multiple `plabel()` options. In this example, the first `plabel()` option assigns the occupation names to all the slices and moves the names 5 units inward. The second `plabel()` option assigns text to the second slice and displays it 5 more units from the center. Because the legend was not needed, we suppressed it with the `legend(off)` option. *Uses nlsw.dta*



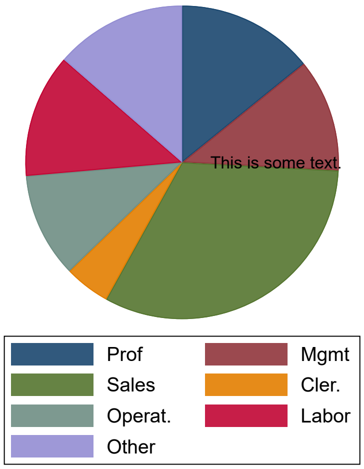
```
graph pie, over(occ7) plabel(_all name, gap(-5)) plabel(_all
percent, gap(5) format("%2.0f")) legend(off)
```

Here we use the `plabel()` option twice to label the slices with the occupation name and with the percentage. We use the `gap()` option to move the names closer to the center by 5 units and move the percentage 5 extra units from the center. *Uses nlsw.dta*



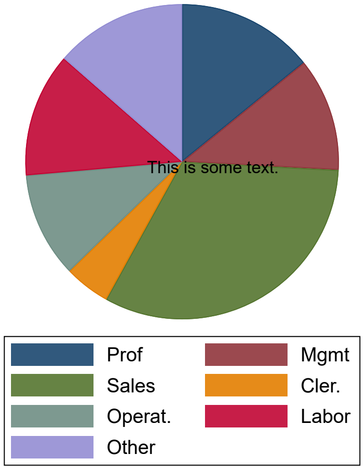
```
graph pie, over(occ7) ptext(0 30 "This is some text.")
```

We can use the `ptext()` (pie text) option to add text to the pie chart. We use polar coordinates to determine the location of the text by specifying the angle and distance from the center. Here the angle is 0, and the distance from the center is 30. *Uses nlsw.dta*



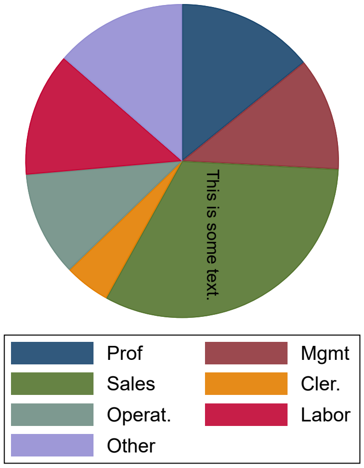
```
graph pie, over(occ7) ptext(-10 10 "This is some text.")
```

Here we choose an angle of -10 (putting it 10 degrees below 0) and a distance of 10 units from the center. The angle determines only the position of the text but not its actual angle of display, which is controlled with the `orientation()` option. See the next example for more details. *Uses nlsw.dta*



```
graph pie, over(occ7) ptext(-10 10 "This is some text.",
orientation(rvertical) placement(s))
```

Now we make the text reverse vertical and start the text from the south of the given coordinates. For more information on these kinds of textbox options, see Options : Textboxes (section 8.11). *Uses nlsw.dta*

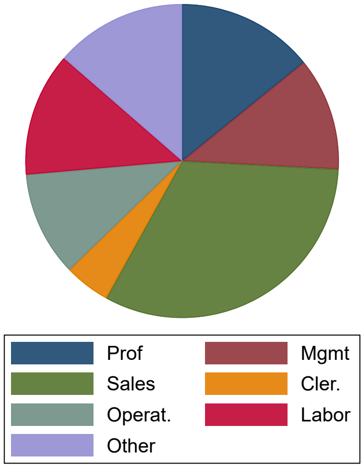

This is some text.
Prof
Mgmt
Sales
Cler.
Operat.
Labor
Other

7.5 Controlling the legend

This section illustrates some of the options that we can use to control the display of the legend with pie charts. Although this section illustrates the use of the legend, it emphasizes options that may be particularly useful with pie charts. See Options : Legend (section 8.9) for more details about the legend; those details apply well to pie charts, even if the examples use other kinds of graphs. Also, see `help legend_options` for more details.

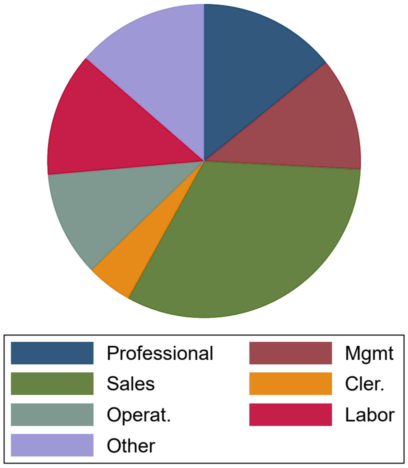
```
graph pie, over(occ7)
```

Consider this pie chart showing the frequencies of women in seven occupational categories. *Uses nlsw.dta*



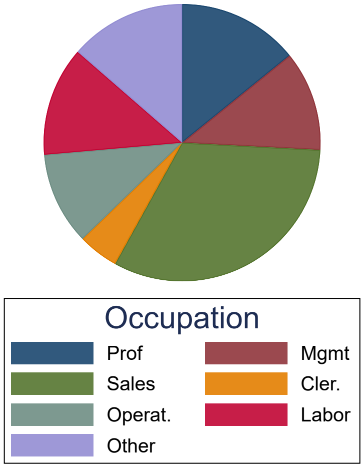
```
graph pie, over(occ7) legend(label(1 "Professional"))
```

We use the `legend(label())` option to change the label for the first occupation. *Uses nlsw.dta*



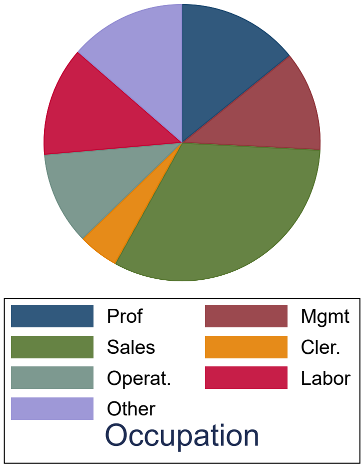
```
graph pie, over(occ7) legend(title(Occupation))
```

The `title()` option within the `legend()` option adds a title to the legend. We can also use `subtitle()`, `note()`, and `caption()` options, much as we would for adding titles to a graph; see **Standard options : Titles** (section 9.1) for more details. *Uses nlsw.dta*



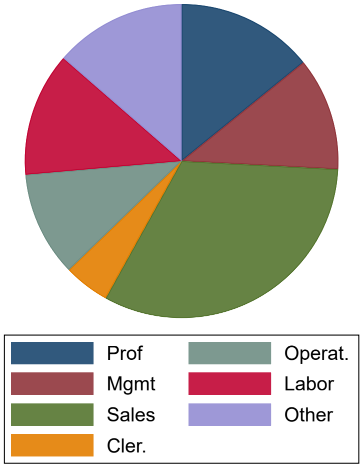
```
graph pie, over(occ7) legend(title(Occupation, position(6)))
```

We can use the `position()` option within the `title()` option to control the position of the title. Here we put the title in the six o'clock position, placing it at the bottom of the legend. *Uses nlsw.dta*



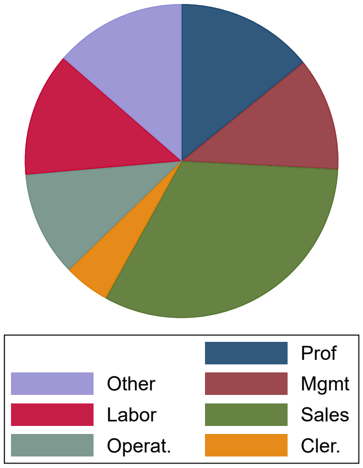
```
graph pie, over(occ7) legend(colfirst)
```

We can use the `legend(colfirst)` option to order the items in the legend by columns instead of rows. *Uses nlsw.dta*



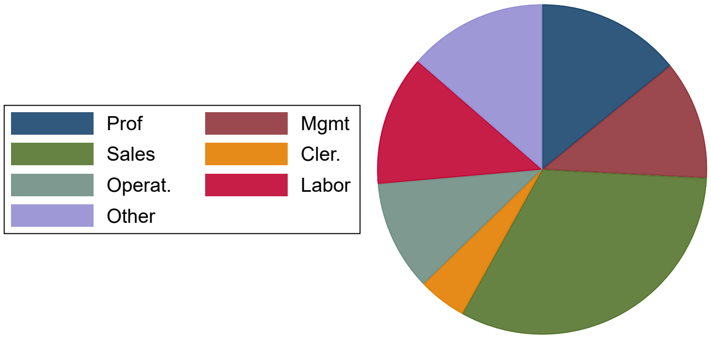
```
graph pie, over(occ7) legend(colfirst order(7 6 5 1 2 3 4)
holes(1))
```

The pie wedges rotate clockwise; here we make the items within the legend rotate in a similar clockwise fashion, starting from the top right. The `order()` option puts the items in the legend in a clockwise order, and the `holes(1)` option leaves the first position empty. *Uses nlsw.dta*



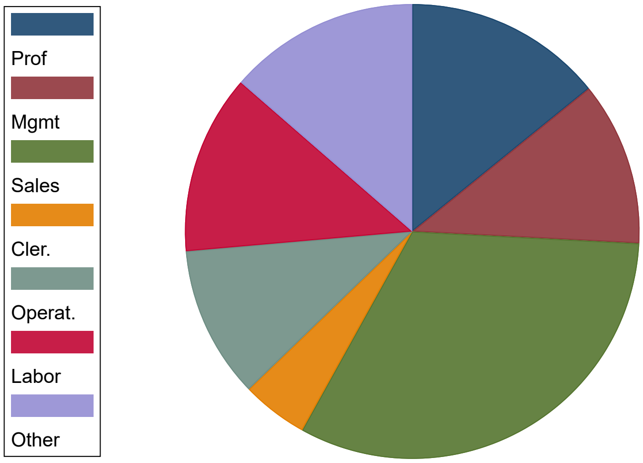
```
graph pie, over(occ7) legend(position(9))
```

Here we put the legend at the nine o’clock position, placing it at the left of the chart. *Uses nlsw.dta*



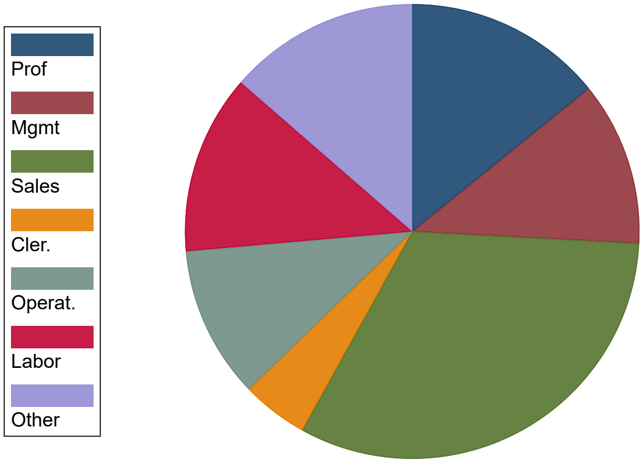
```
graph pie, over(occ7) legend(position(9) cols(1) stack)
```

Adding the `cols(1)` and `stack` options makes the legend display in one column with the symbol stacked above the descriptive text. *Uses nlsw.dta*



```
graph pie, over(occ7) scheme(vg_lgndc)
```

Here we use the `scheme(vg_lgndc)` option. By specifying this scheme, the legend is positioned to the left in one column with the symbol stacked above the description. *Uses nlsw.dta*

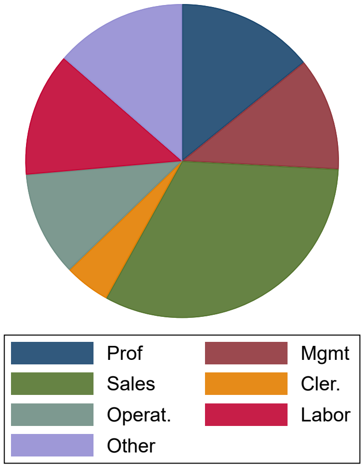


7.6 Graphing by groups

This section describes the use of the `by()` option with pie charts, focusing on features that are specifically relevant to pie charts. For more details, see Options : By (section 8.8) and `help by_option`.

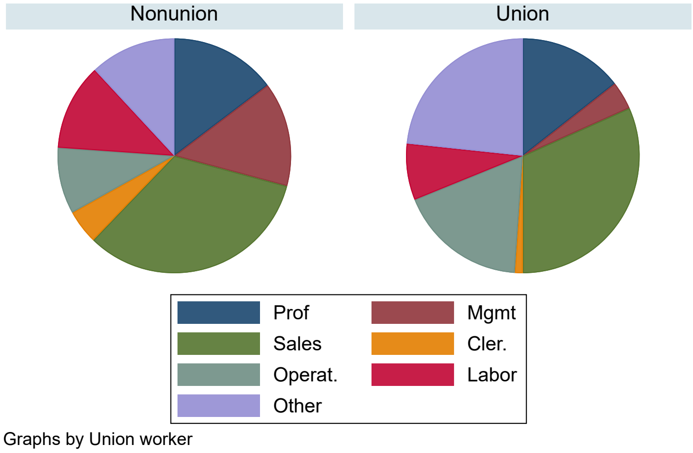
```
graph pie, over(occ7)
```

Here we see a basic pie chart showing the distribution of occupations. *Uses nlsw.dta*



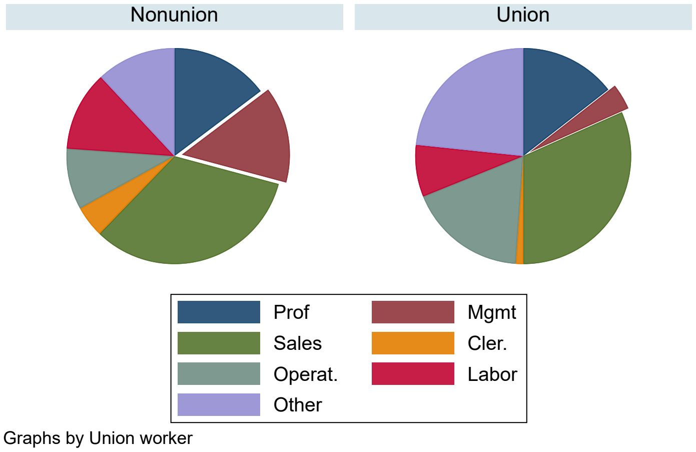
```
graph pie, over(occ7) by(union)
```

In this graph, the occupations are broken down by whether one belongs to a union. *Uses nlsw.dta*



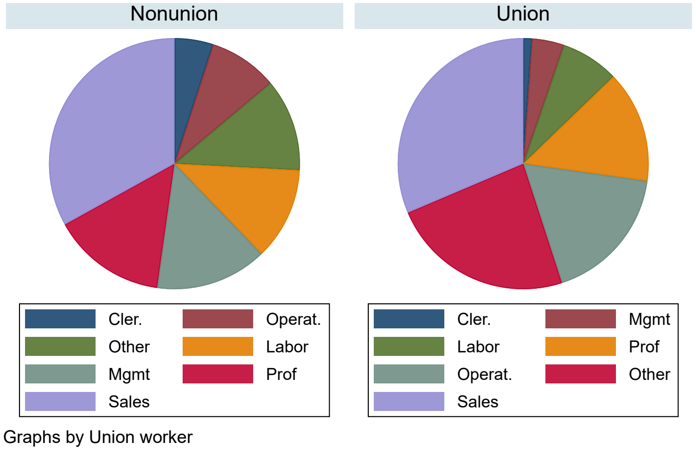
```
graph pie, over(occ7) by(union) pie(2, explode)
```

If we add the `pie(2, explode)` option, the second slice is exploded in both graphs. *Uses nlsw.dta*



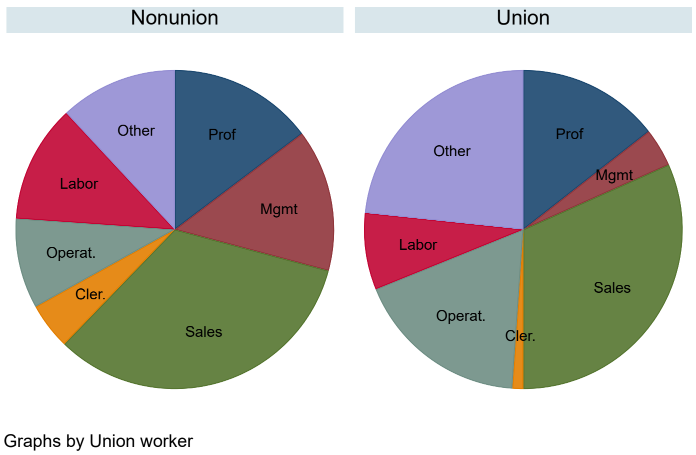
```
graph pie, over(occ7) by(union) sort
```

Here we sort the slices from least frequent to most frequent. A separate legend is shown for each chart. This is because we can order the slices differently in the two different graphs when sorted. Thus, when we use the `sort` option for pie charts, Stata shows two separate legends to assure proper labeling of the slices. *Uses nlsw.dta*

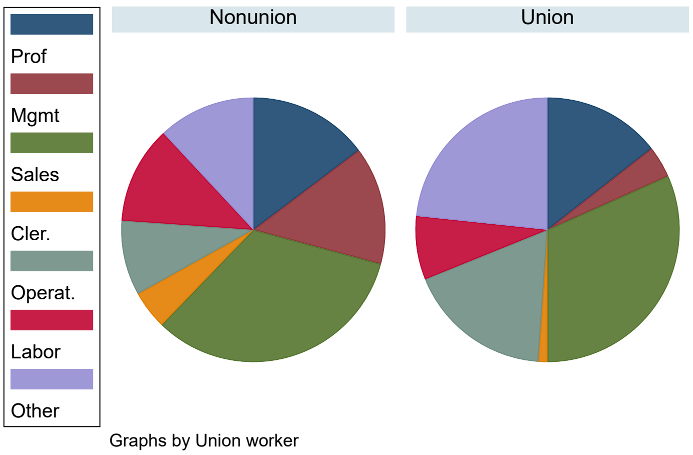


```
graph pie, over(occ7) by(union, legend(off)) plabel(_all
name)
```

In this example, we add the `plabel()` option to label the inside of each slice with the name of the slice, so the legend is no longer needed. We suppress the legend with the `legend(off)` option, which is placed within the `by()` option because it, in a way, determines the placement of the legend by turning it off. *Uses nlsw.dta*



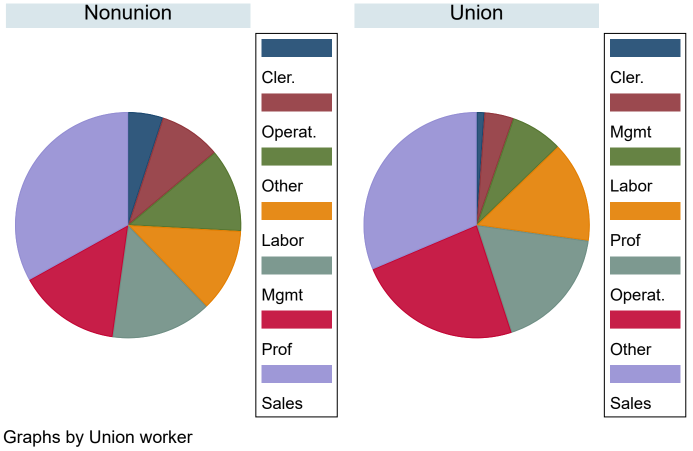
```
graph pie, over(occ7) by(union, legend(pos(9)))
legend(cols(1) stack)
```



Here we place the legend to the left by using the `legend(pos(9))` option. This option is contained within the `by()` option because it alters the position of the legend. We also make the legend one column with the legend symbols and labels stacked with the `legend(cols(1) stack)` option. This option is outside the `by()` option because it does not determine the position of the legend. *Uses nlsw.dta*

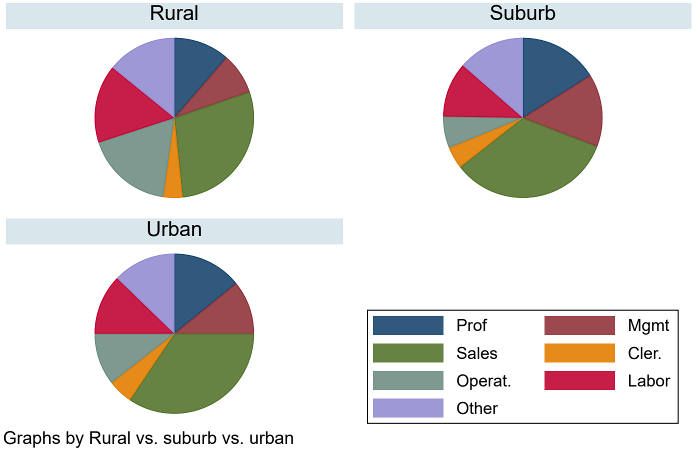
```
graph pie, over(occ7) by(union) legend(pos(3) cols(1) stack)
sort
```

This example is similar to the previous one, but we have added the `sort` option. When we add the `sort` option, we need to move the `pos()` option from within the `by()` option to outside the `by()` option. This is an exception to the general rule that legend options that control the position of the legend are placed within the `by()` option. Here we get the legends that we desire, each to the right of the pie. *Uses nlsw.dta*



```
graph pie, over(occ7) by(urban3, legend(at(4)))
```

Here we break down occupation by a three-level variable, leaving a fourth position open. We can specify the `legend(at(4))` option within the `by()` option to place the legend in the space in the fourth position. *Uses nlsw.dta*



Chapter 8
Options available for most graphs

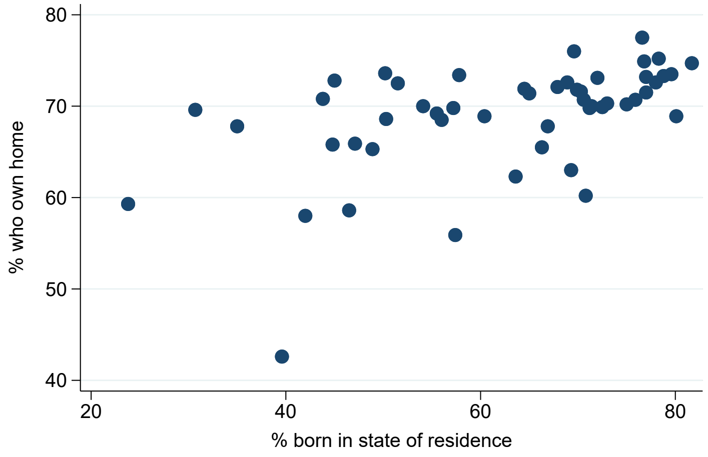
This chapter discusses options that are used in many, but not all, kinds of graphs in Stata, as compared with the Standard options (section 9) chapter, which covers options that are standard in all Stata graphs. This chapter goes into greater detail about how to use these options to customize your graphs. As you can see from the *Visual Table of Contents* at the right, this chapter covers markers, how to connect points and markers, axis titles, axis labels, axis scales, axis selection, the `by()` option, the legend, added text, textboxes, and options for controlling the display of text. For further details, the examples will frequently refer to Styles (section 10) and `help graph`.

8.1 Changing the look of markers

This section looks at options for controlling markers. Although the examples in this section focus on `twoway scatter`, these options apply to any graph where you have markers and can control them. This section will show how to change the marker symbol, size, and color (both fill and outline color). For more information, see `help marker_options`.

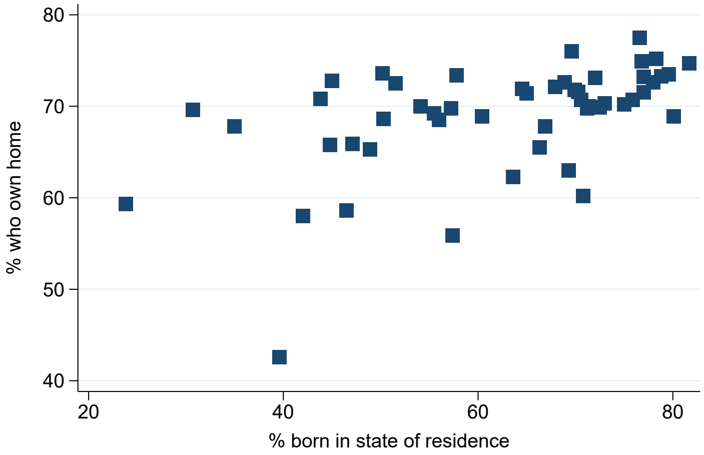
```
twoway scatter ownhome borninstate
```

Consider this scatterplot showing the relationship between the percentage of people in a state who own their home and the percentage of people born in their state of residence. The markers in this plot are filled circles. *Uses allstates.dta*



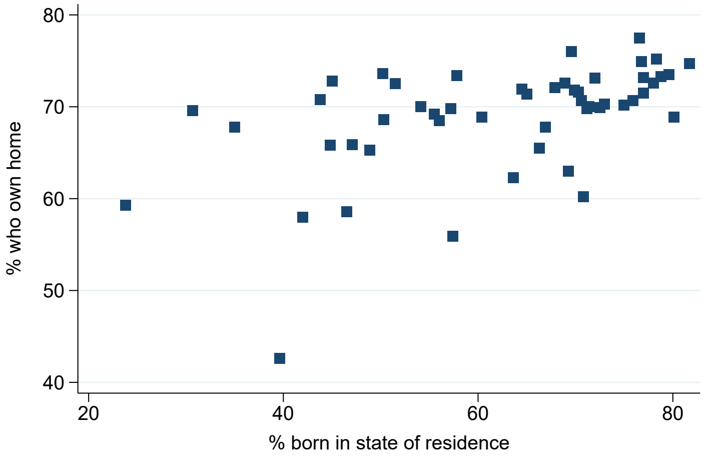
```
twoway scatter ownhome borninstate, msymbol(S)
```

We can control the shape of the marker with the `msymbol()` (marker symbol) option. Here we make the symbols large squares. See **Styles : Symbols** (section 10.11) for more information about symbols. *Uses allstates.dta*



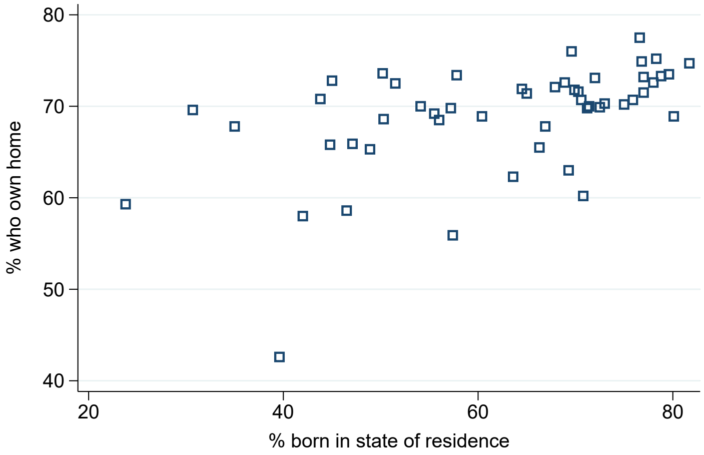
```
twoway scatter ownhome borninstate, msymbol(s)
```

Specifying `msymbol(s)`, which uses a lowercase `s`, displays smaller squares. *Uses allstates.dta*



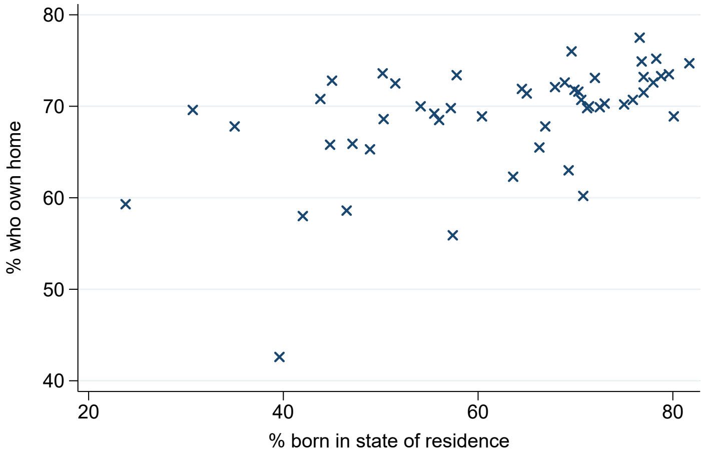
```
twoway scatter ownhome borninstate, msymbol(sh)
```

We can append an `h` (that is, `msymbol(sh)`) to yield small, hollow squares. In addition to choosing `S` for larger squares and `s` for small squares, we can specify `D` (large diamond), `T` (large triangle), and `O` (large circles). We can specify lowercase letters to get smaller versions of these symbols and append `h` for hollow versions. *Uses allstates.dta*



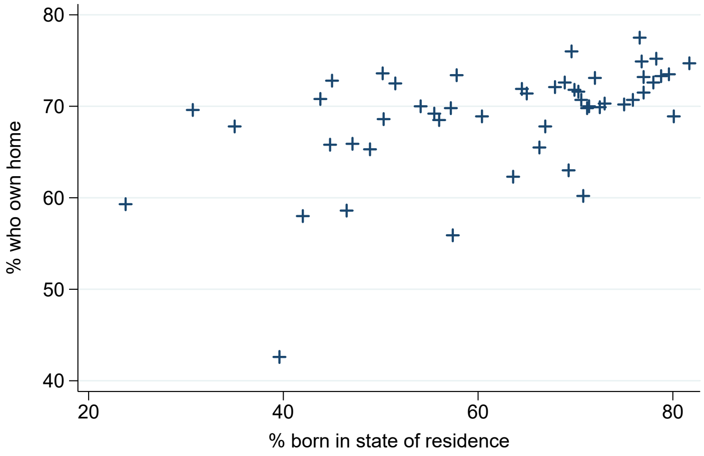
```
twoway scatter ownhome borninstate, msymbol(X)
```

We can also specify `msymbol(X)` to use a large X for the markers. A lowercase `x` could be used for smaller markers. We cannot append an `h` because we cannot make a hollow X. *Uses allstates.dta*



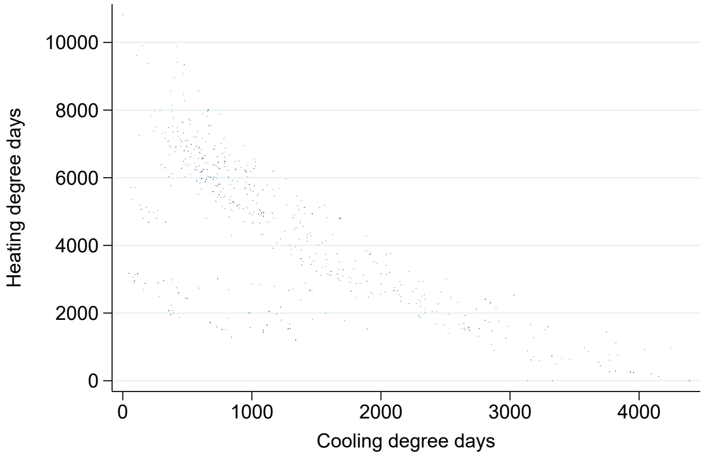
```
twoway scatter ownhome borninstate, msymbol(+)
```

Specifying `msymbol(+)` yields a plus sign for the markers. We cannot make the plus signs hollow or make a smaller version. *Uses allstates.dta*



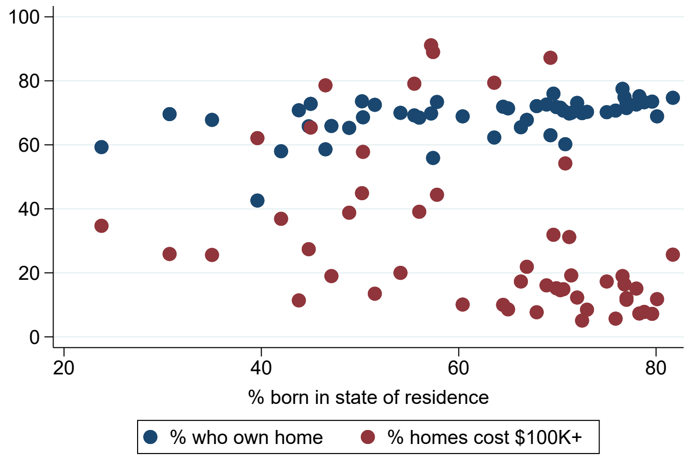
```
twoway scatter heatdd cooldd, msymbol(p)
```

Here we switch to the `citytemp` data file to illustrate the use of the `msymbol(p)` option to plot small points. Although each point is hard to see because it is so small, we can see the overall pattern of the data because of the many points and the strong trend in the data. See **Styles : Symbols** (section 10.11) for more information about symbols. *Uses citytemp.dta*



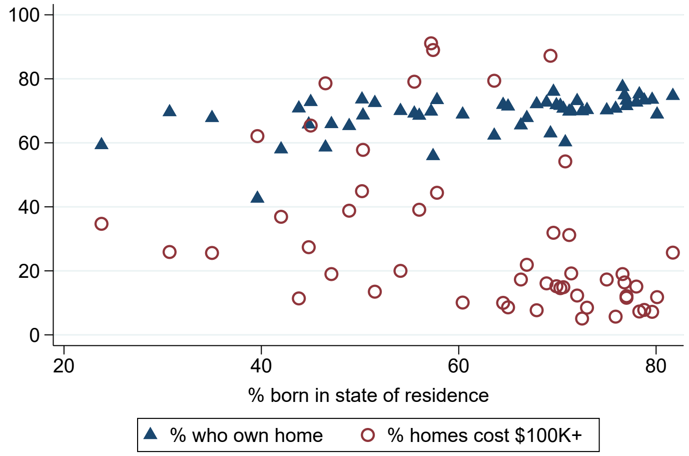
```
twoway scatter ownhome propval100 borninstate
```

Aside from aesthetics, choosing different marker symbols is useful to differentiate multiple markers displayed in the same plot. In this example, we plot two y variables, and Stata displays both as solid circles, differing in color. *Uses allstates.dta*



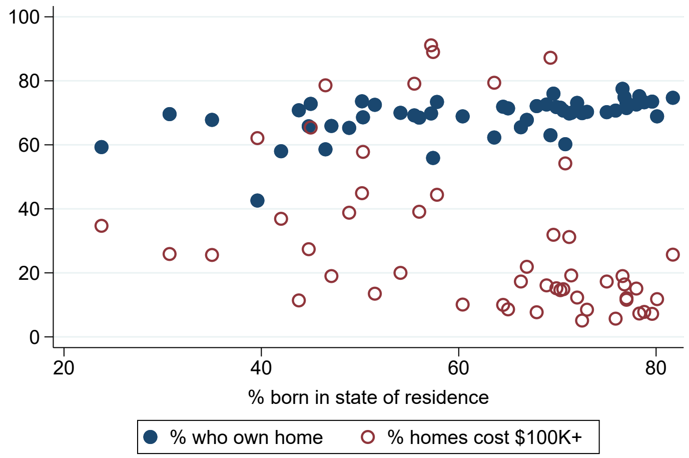
```
twoway scatter ownhome propval100 borninstate, msymbol(t Oh)
```

To further differentiate the symbols, we add the `msymbol(t Oh)` option to control both markers. Here we make the first marker a small triangle and the second a larger hollow circle. *Uses allstates.dta*



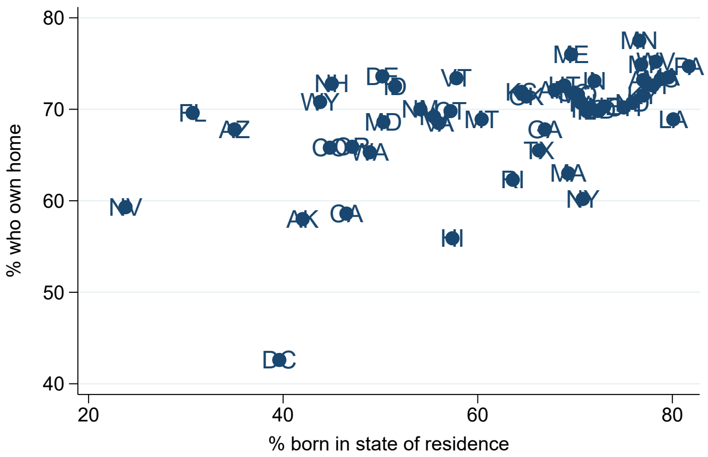
```
twoway scatter ownhome propval100 borninstate, msymbol(. Oh)
```

By using the `msymbol(. Oh)` option, we can leave the first symbol unchanged (as indicated by the dot) and change the second symbol to a hollow circle. We might think that the dot indicates a small point but that is indicated by the `p` option. *Uses allstates.dta*



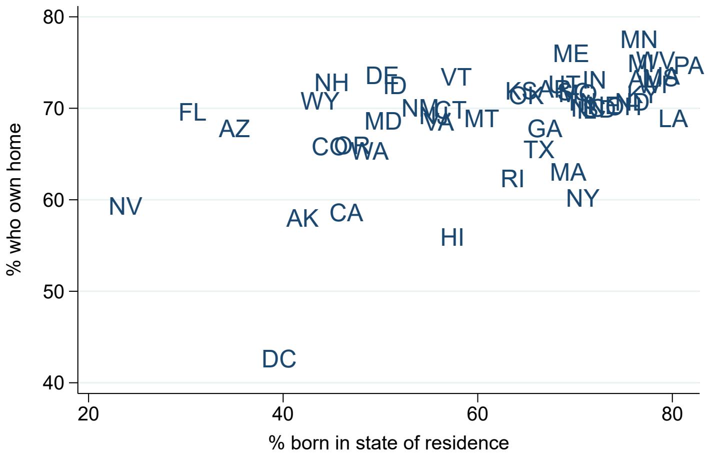
```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabpos(center)
```

One last marker symbol is `i` for invisible, allowing us to hide the marker symbol. Here we use the `mlabel(stateab)` (marker label) option to display a marker label with the state abbreviation for each observation and the `mlabpos(center)` (marker label position) option to center the marker label. However, the marker symbol (the circle) and the marker label (the abbreviation) are on top of each other. *Uses allstates.dta*



```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabpos(center) msymbol(i)
```

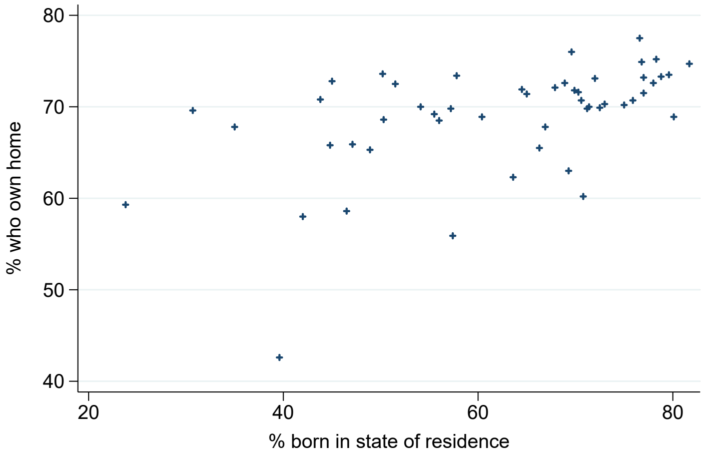
If we use `msymbol(i)` to make the marker symbol invisible, the marker label (the state abbreviation) can be displayed without being obscured by the marker symbol. See Styles : Symbols (section 10.11) for more information about symbols. *Uses allstates.dta*



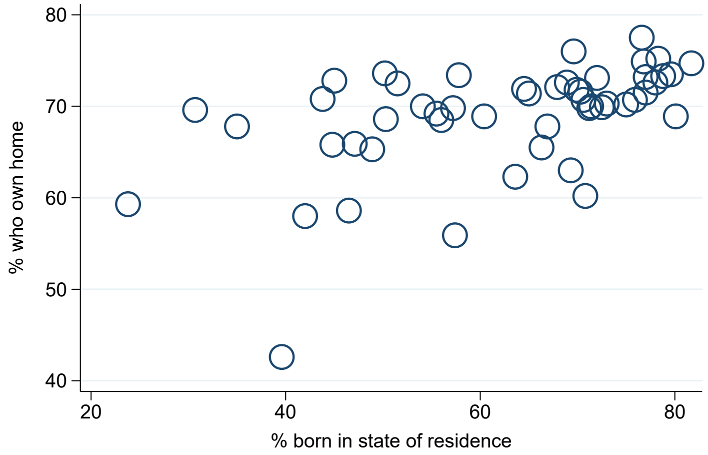
So far, we have seen that we can use the `msymbol()` option to control the marker symbol and, to a certain extent, to control the marker size (for example, using `O` yields large circles and using `o` yields small circles). As the following examples show, we can use the `msize()` option to exert more flexible control over the size of the markers.

```
twoway scatter ownhome borninstate, msymbol(+) msize(small)
```

Here we use the `msize()` (marker size) option to control the size of the marker symbol, making the marker symbol small. Values we could have chosen include `vtiny`, `tiny`, `vsmall`, `small`, `medsmall`, `medium`, `medlarge`, `large`, `vlarge`, `huge`, `vhuge`, and `ehuge`. See Styles : Markersize (section 10.9) for more details (including sizing using points, inches, and centimeters). *Uses allstates.dta*



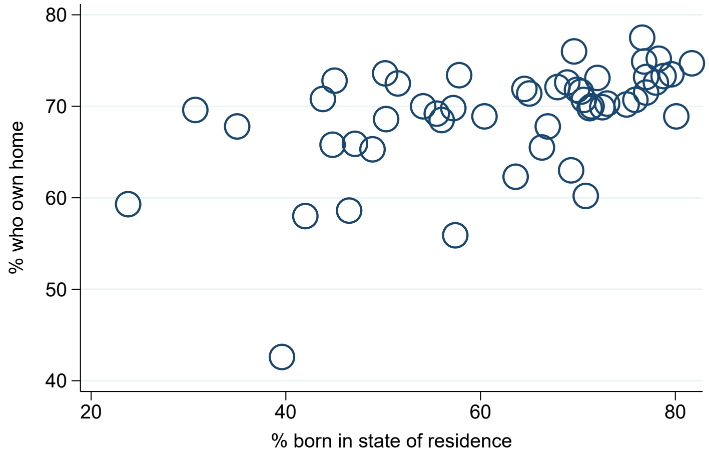
```
twoway scatter ownhome borninstate, msymbol(Oh) msize(*2)
```



We can specify the sizes as multiples of the original size of the marker. In this example, we make the markers twice their original size by specifying `msize(*2)`. Specifying a value less than one reduces the marker size; for example, `msize(*.5)` would make the marker half its normal size. See **Styles : Markersize** (section 10.9) for more details (including sizing using points, inches, and centimeters). *Uses allstates.dta*

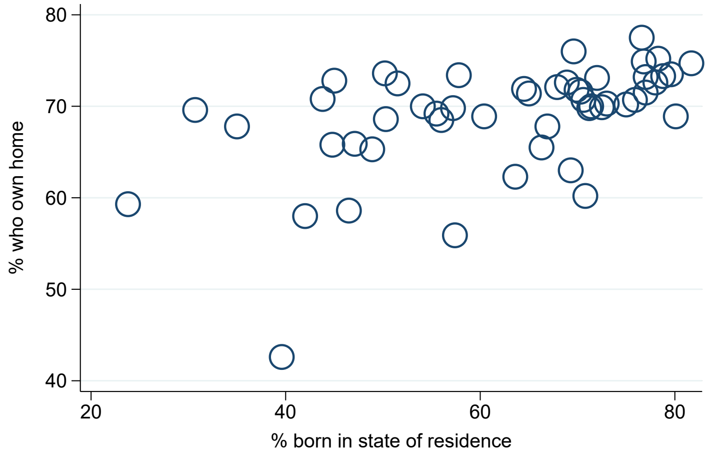
```
twoway scatter ownhome borninstate, msymbol(Oh) msize(5rs)
```

We can specify the sizes as a multiple of the graph size (the smaller of the height and width). For this graph (which is 3 inches by 2 inches), the smaller size is 2. Specifying `msize(5rs)` translates to 5% of 2 inches, or 0.1 inch. See **Styles : Markersize** (section 10.9) for more details (including sizing using points, inches, and centimeters). *Uses allstates.dta*



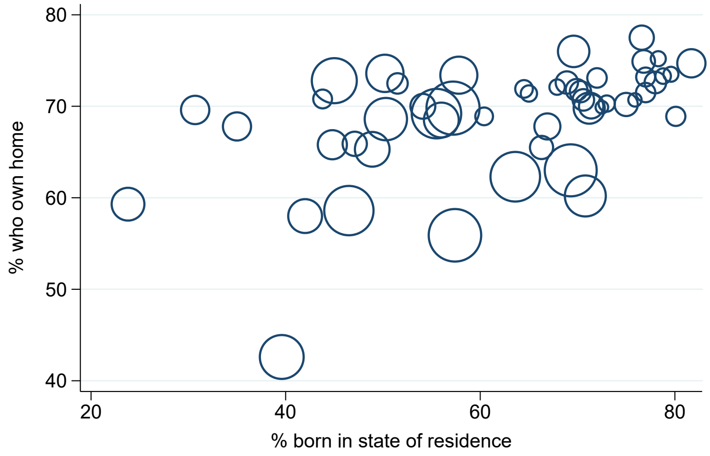
```
twoway scatter ownhome borninstate, msymbol(Oh) msize(7pt)
```

You can also specify the marker size using absolute units, either in points, inches, or centimeters. Specifying `msize(7pt)` yields markers that are 7 points in size. See Styles : Markersize (section 10.9) for more details (including sizing using points, inches, and centimeters). Also see section 9.3.1 regarding resizing graphs using absolute units. *Uses allstates.dta*



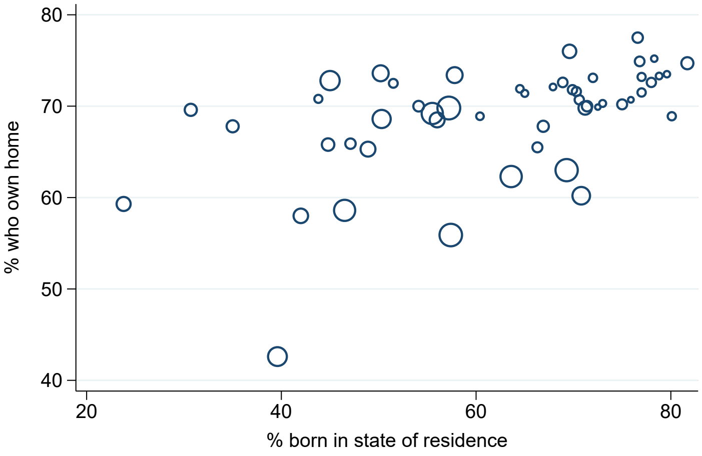
```
twoway scatter ownhome borninstate [aweight=propval100],
msymbol(oh)
```

Stata even allows us to size the symbols on the values of another variable in your data file. This allows us, in a sense, to graph three variables at once. Here we look at the relationship between `borninstate` and `ownhome` and then size the markers on `propval100` by using `[aweight=propval100]`, weighting the markers by `propval100`. *Uses allstates.dta*



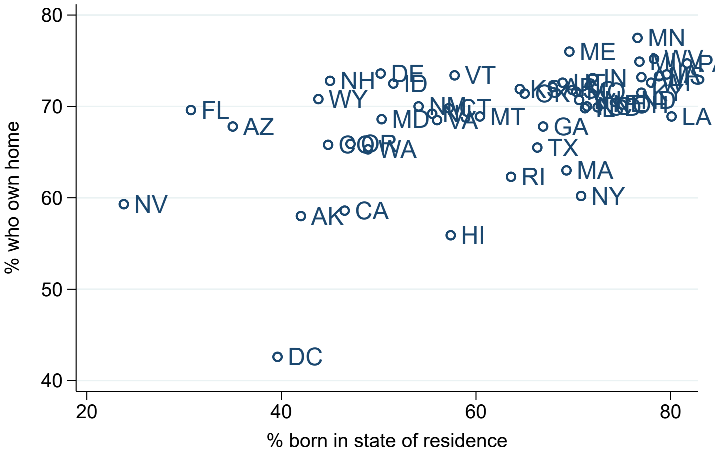
```
twoway scatter ownhome borninstate [aweight=propval100],
msymbol(oh) msize(small)
```

Even if we weight the size of the markers by using `aweight`, we can still control the general size of the markers. Here we make all markers smaller by using the `msize(small)` option. The markers are smaller than they were previously but are still sized according to the value of `propval100`. *Uses allstates.dta*



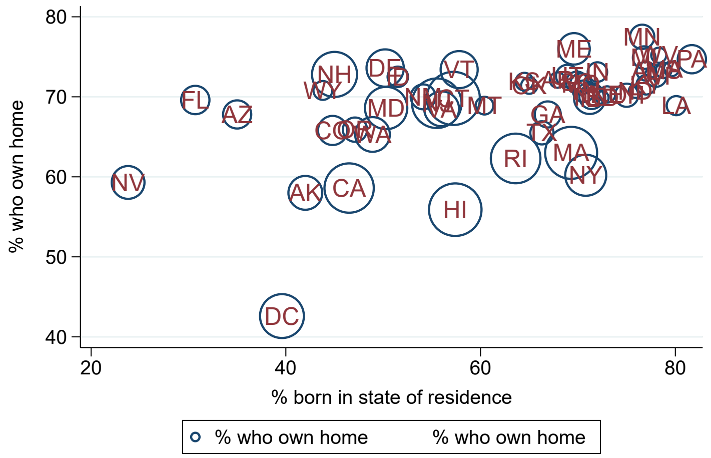
```
twoway scatter ownhome borninstate [aweight=propval100],
msymbol(oh) msize(large) mlabel(stateab)
```

We might even try to graph a fourth variable in the plot by using the `mlabel()` (marker label) option. Here we try to use the `mlabel(stateab)` option to label each marker with the abbreviation of the state. However, when we add the `mlabel()` option, the weights no longer affect the size of the markers. See the following example for a solution to this. *Uses allstates.dta*



```
twoway (scatter ownhome borninstate [aweight=propval100],
msymbol(oh) msize(large)) (scatter ownhome borninstate,
mlabel(stateab) msymbol(i) mlabpos(center))
```

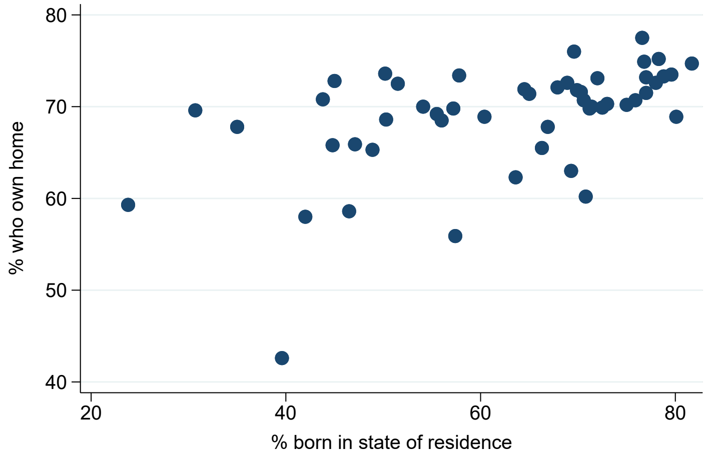
We can solve the problem from the previous example by overlaying a scatterplot that has the symbols weighted by `propval100` with a scatterplot that shows just the marker labels. The second scatterplot uses the `mlabel(stateab) msymbol(i) mlabpos(center)` options to label the markers with the state abbreviation. See **Options : Marker labels** (section 8.2) for more details. *Uses allstates.dta*



Stata also allows us to control the color of the markers. We can control the overall color of the marker: create a solid color or make the inner part of the marker one color (called a fill color) and the outline of the marker a different color. We can also vary the thickness of the outline of the marker. See **Styles : Colors** (section 10.2) for more information about specifying colors, including the different colors you can specify, how to adjust opacity (transparency), how to adjust brightness, and options for specifying colors using technical methods (such as RGB, CMYK, and HSV/HSL/HSB).

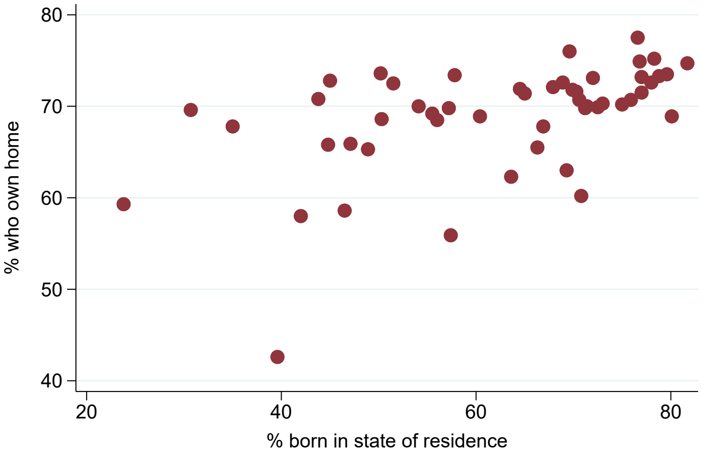
```
twoway scatter ownhome borninstate
```

For this first example, let’s create a scatterplot using the default settings. Note that the markers are displayed in blue—more specifically, they are displayed using the color Stata calls `navy`. *Uses allstates.dta*



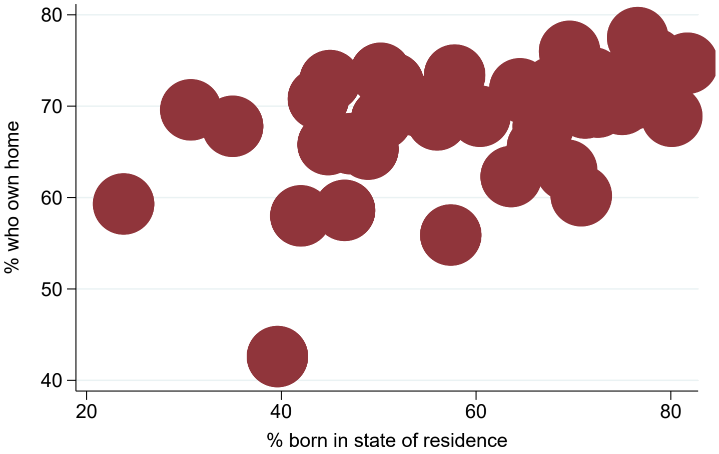
```
twoway scatter ownhome borninstate, mcolor(maroon)
```

We can use the `mcolor()` (marker color) option to control the color of the markers. Here we make the markers maroon by using the `mcolor(maroon)` option. See Styles : Colors (section 10.2) for more information about specifying colors. *Uses allstates.dta*



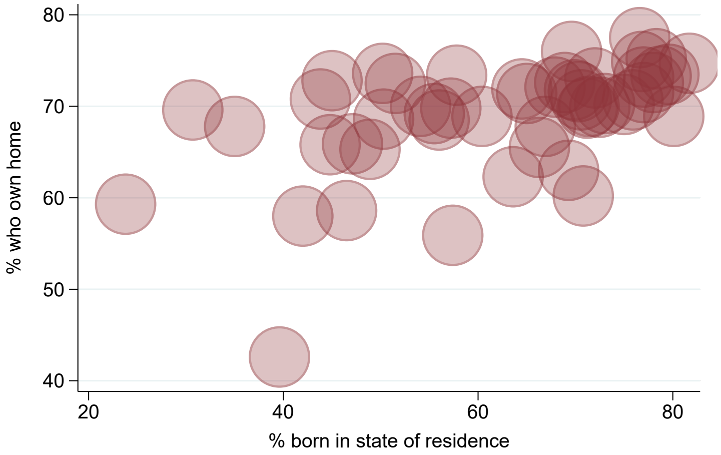
```
twoway scatter ownhome borninstate, msize(*5) mcolor(maroon)
```

Some of the options that I will illustrate are hard to see using normal sized markers on the small graphs shown in this book. For pedagogical purposes, I am going to use the `msize(*5)` option to make the marker labels five times their normal size. This will make it easier to see the impact of some of the options shown in the following examples. *Uses allstates.dta*



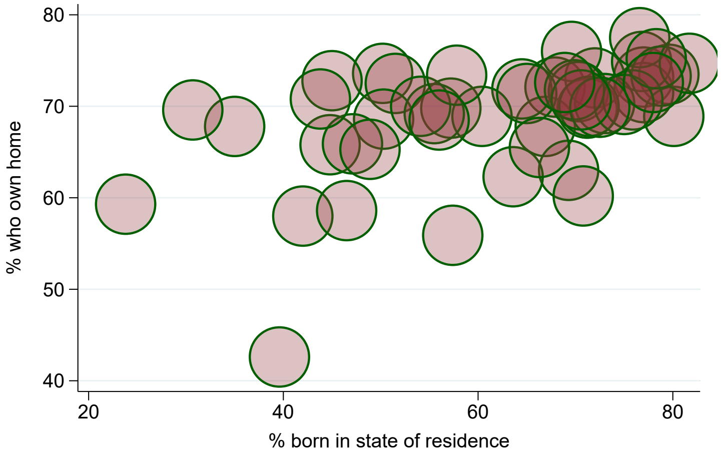
```
twoway scatter ownhome borninstate, msize(*5)
mcolor(maroon%30)
```

By specifying `mcolor(maroon%30)`, the color of the markers are `maroon`, and they are displayed using 30% opacity (or, if you prefer, 70% transparency). Note how the markers are drawn with a line that outlines the shape of the marker. The next examples will illustrate how to control the color and thickness of that line. *Uses allstates.dta*



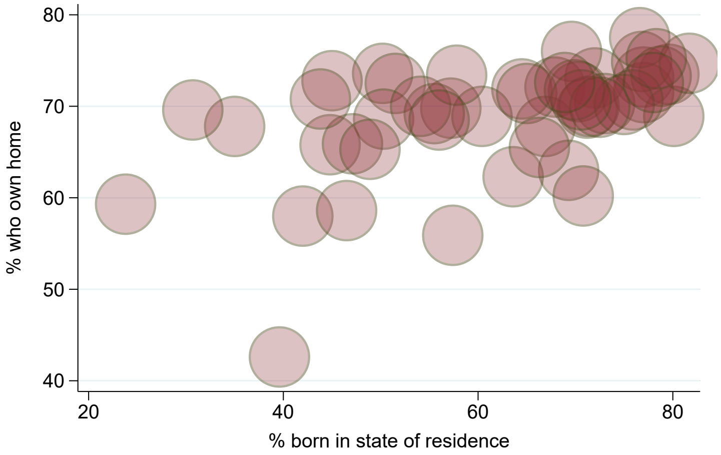
```
twoway scatter ownhome borninstate, msize(*5)
mfcolor(maroon%30) mlcolor(dkgreen)
```

We can separately control the fill color (inside color) and outline color with the `mfcolor()` (marker fill color) and `mlcolor()` (marker line color) options, respectively. Here we make the fill color `maroon` with 30% opacity specifying `mfcolor(maroon%30)` and the line color dark green by specifying `mlcolor(dkgreen)`. *Uses allstates.dta*



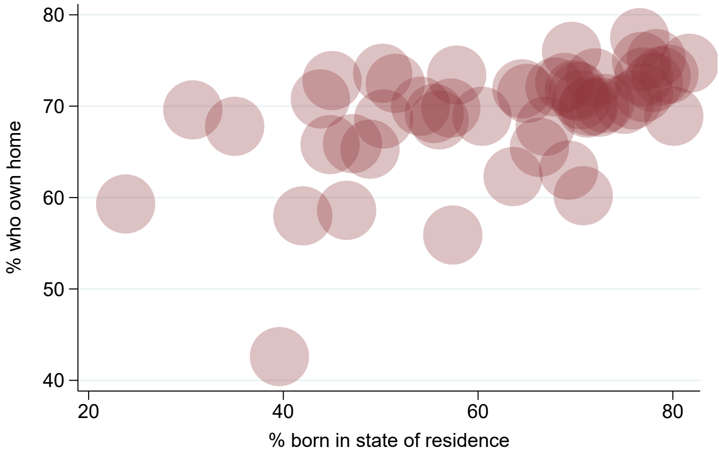
```
twoway scatter ownhome borninstate, msize(*5)
mfcolor(maroon%30) mlcolor(dkgreen%20)
```

You can modify the opacity of the outline color if you like. This example is like the previous example, except that by specifying `mlcolor(dkgreen%20)`, the outline color is displayed with 20% opacity. *Uses allstates.dta*



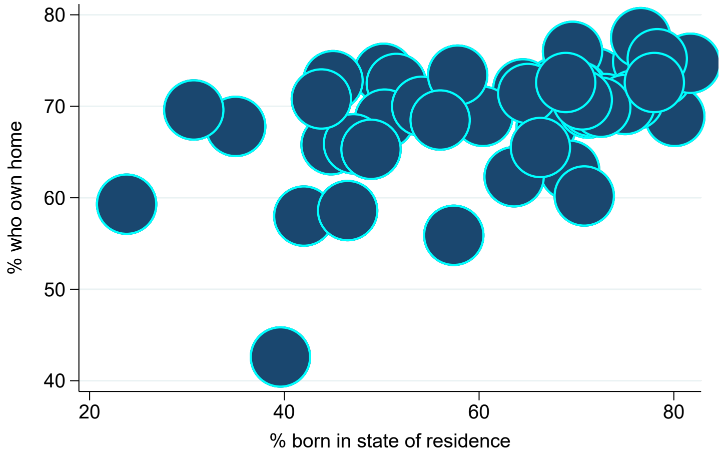
```
twoway scatter ownhome borninstate, msize(*5)
mfcolor(maroon%30) mlcolor(none)
```

In this example, we specified `mlcolor(none)`, which made the outline color invisible; in other words, no outline is drawn. As you can see, the markers are displayed using `maroon` with 30% opacity. *Uses allstates.dta*



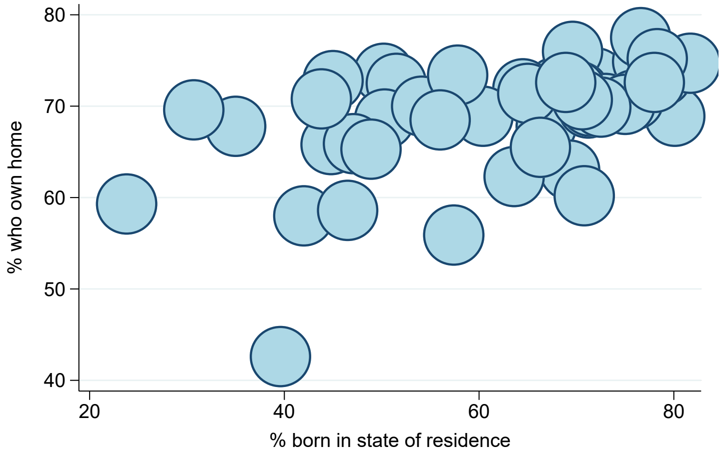
```
twoway scatter ownhome borninstate, msize(*5) mlcolor(cyan)
```

We can change just the line color surrounding the marker with the `mlcolor()` option. Here we specify `mlcolor(cyan)` to make the line surrounding the markers `cyan`. Note how the markers are filled in navy blue, the default marker fill color. *Uses allstates.dta*

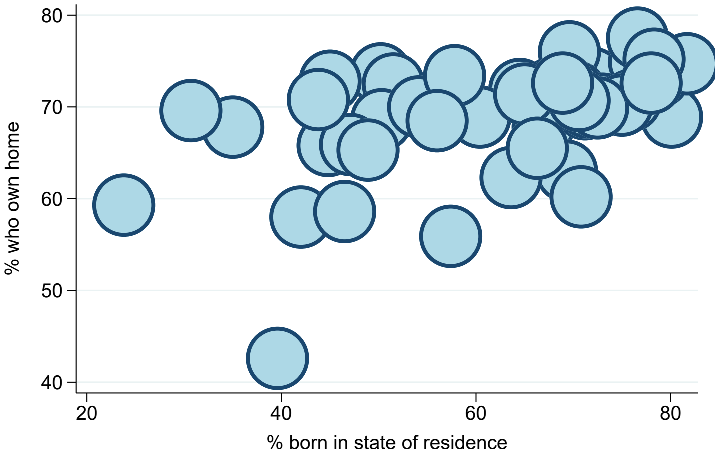


```
twoway scatter ownhome borninstate, msize(*5) mfcolor(ltblue)
```

We can also separately control the fill color using the `mfcolor()` option. Here we choose `mfcolor(ltblue)` to make the fill color light blue. Note how the line color surrounding the marker is displayed using `navy`, the default. *Uses allstates.dta*



```
twoway scatter ownhome borninstate, msize(*5) mfcolor(ltblue)
mlwidth(thick)
```

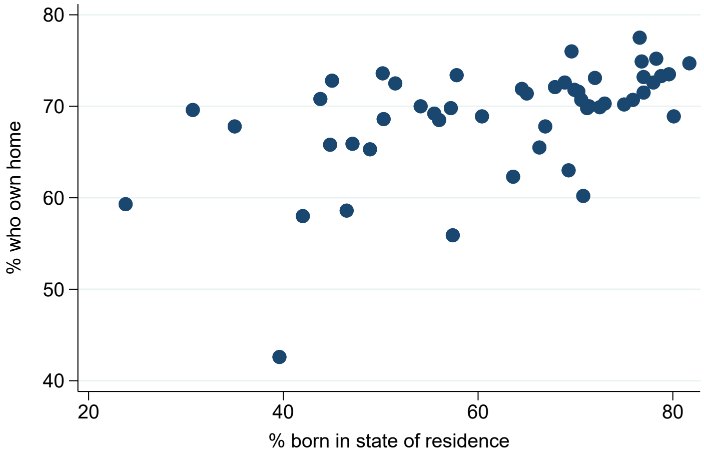


Adding the `mlwidth(thick)` (marker line width) option makes the line around the marker thick. We can also indicate the thickness as a multiple of the original thickness; for example, `mlwidth(*3)` indicates the line should be three times as thick as it would normally be. We can also specify thickness using points. See Styles : Linewidth (section 10.7) for more details. *Uses allstates.dta*

So far, these examples have focused on controlling the individual elements of markers: the symbol, color, size, fill color, line color, and so forth. There is another way to change the appearance of a marker: by specifying a marker style. The marker style controls all these attributes at once, and sometimes it can be more efficient to use a marker style to control the elements individually, as we will see in the following examples.

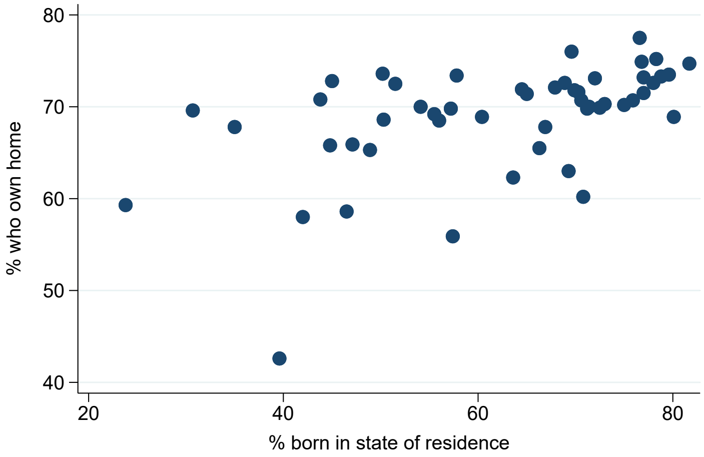
```
twoway scatter ownhome borninstate
```

The marker styles are named/numbered `p1–p15`. The markers here are displayed using the `p1` style because we are plotting only one y variable and have not specified a marker style. *Uses allstates.dta*



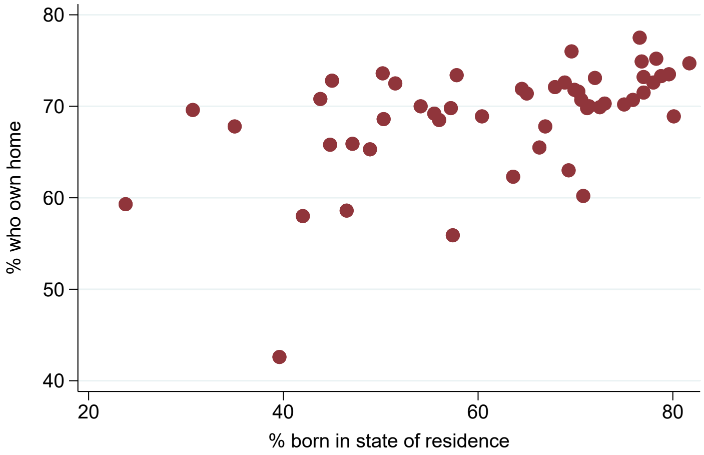
```
twoway scatter ownhome borninstate, mstyle(p1)
```

Here we explicitly select the default marker style by using the `mstyle(p1)` (marker style) option, and the markers look identical to the previous graph. *Uses allstates.dta*



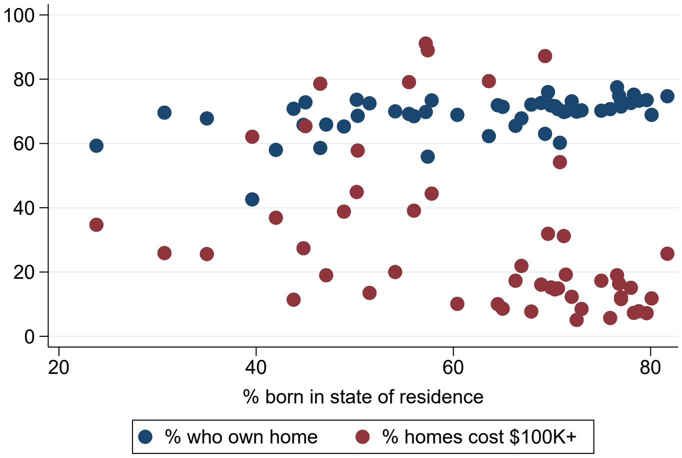
```
twoway scatter ownhome borninstate, mstyle(p2)
```

Here we use `mstyle(p2)` to explicitly select the `p2` style for displaying the markers, and now the markers are different in size, shape, and color. The markers are now larger diamonds that are a middle-level gray color. *Uses allstates.dta*



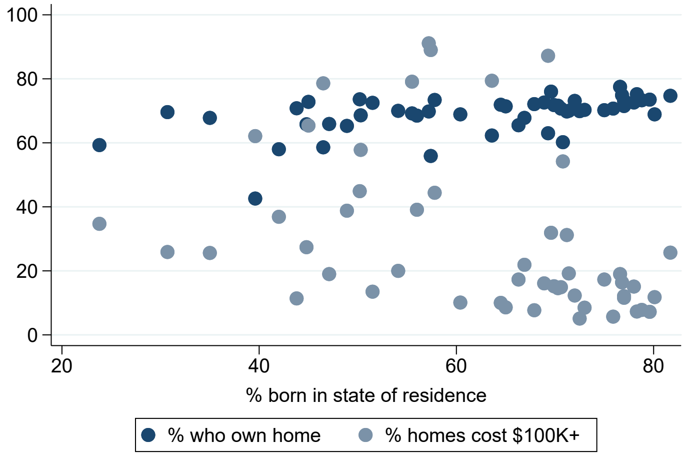
```
twoway scatter ownhome propval100 borninstate
```

Now, when we plot two variables, the first variable is plotted with the `p1` style and the second variable is plotted with the `p2` style. We would have gotten the same result if we had specified the option `mstyle(p1 p2)`. *Uses allstates.dta*

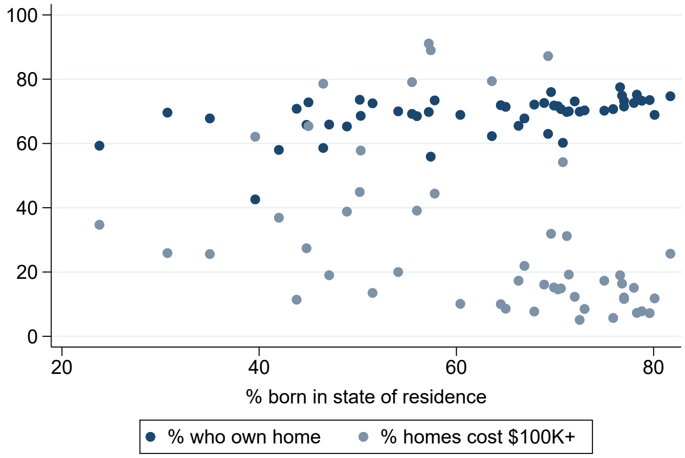


```
twoway scatter ownhome propval100 borninstate, mstyle(p1 p10)
```

Here we use the `mstyle(p1 p10)` option to request that the first variable be plotted with the `p1` style and the second be plotted with the `p10` style. A style is just a starting point, and we can use more options to modify the markers to suit our taste. *Uses allstates.dta*



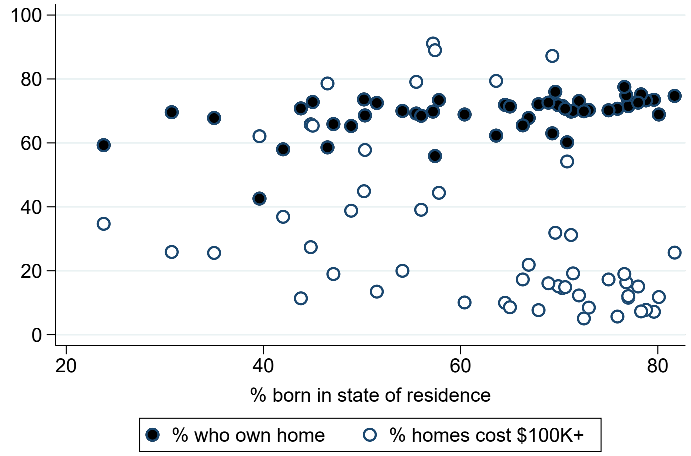
```
twoway scatter ownhome propval100 borninstate, mstyle(p1 p10)
msize(. medium)
```



Say that in the previous graph you wanted medium-sized triangles. We can add the `msize(. medium)` option to control the size of the second marker, leaving the first unchanged. So, even though a style chooses a number of characteristics for the markers, we can override them. *Uses allstates.dta*

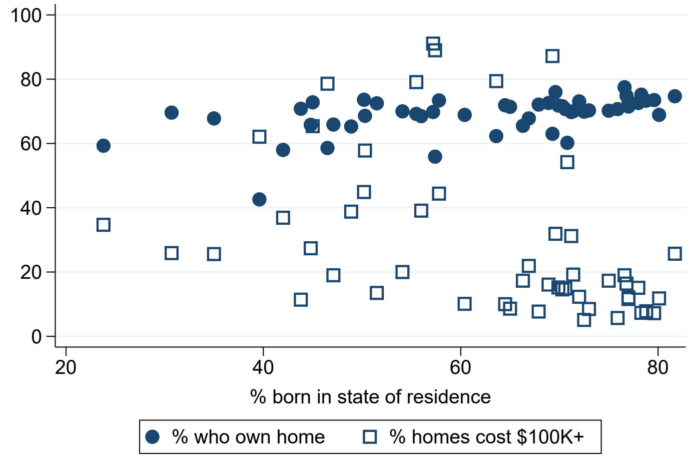
```
twoway scatter ownhome propval100 borninstate, mstyle(p1 p1)
mfcolor(. white)
```

In this example, we use the `p1` style for both the first and second markers, creating small, dark gray, filled circles. Adding the `mfcolor(. white)` option makes the fill color as-is for the first variable and white for the second variable. *Uses allstates.dta*



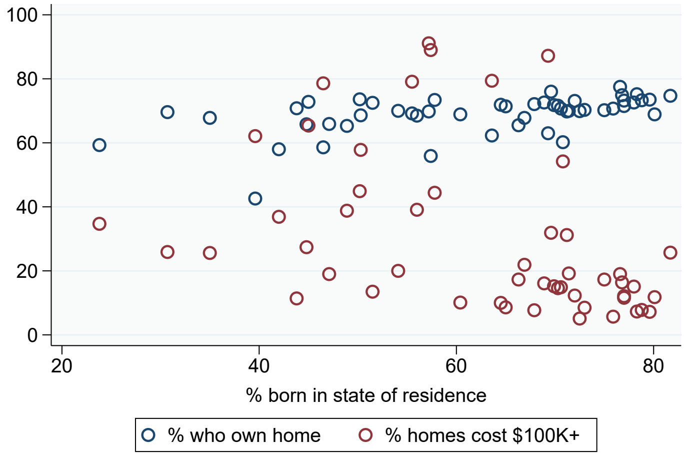
```
twoway scatter ownhome propval100 borninstate, msymbol(. Sh)
scheme(vg_samec)
```

Another strategy for controlling the marker symbols is choosing or creating a scheme. The `vg_samec` scheme makes all markers solid, blue circles, allowing you to customize them all from a common base. Adding the `msymbol(. Sh)` option makes the second symbol hollow squares. *Uses allstates.dta*



```
twoway scatter ownhome propval100 borninstate,
scheme(vg_hollowc)
```

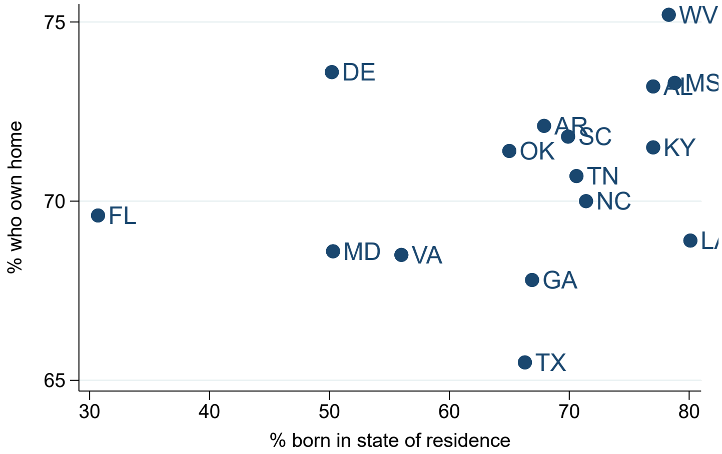
A number of these markers are clustered together, sometimes creating overlap among solidly filled markers. By using `scheme(vg_hollowc)` scheme, the fill color for markers is invisible. As you can see, using an invisible fill can aid in seeing markers that are clumped together or nearly overlapping. *Uses allstates.dta*


100
80
60
40
20
0
20
40
60
80
% born in state of residence
○ % who own home
○ % homes cost $100K+

8.2 Creating and controlling marker labels

This section looks at the details of using marker labels. We can use marker labels to identify the markers with `graph twoway`, but we can also use them with other types of graphs, such as `graph matrix` and `graph box`, affecting the outside values. We can even use marker labels instead of markers. For more information, see `help marker_label_options`. This section will use `allstates3.dta`, which keeps the states that are in the South, that is, if `region` is equal to 3.

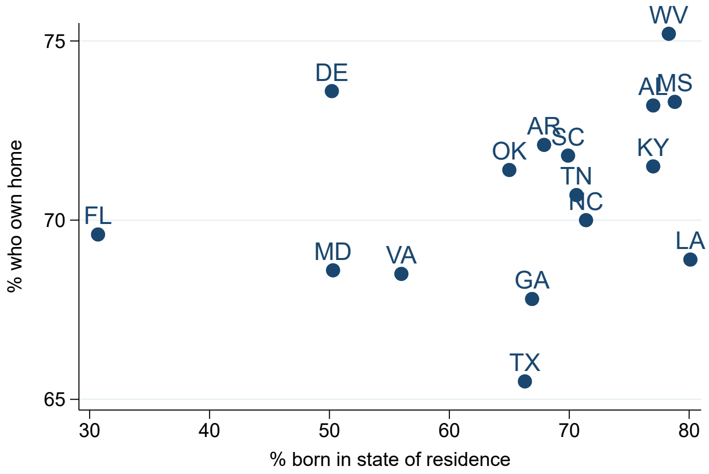
```
twoway scatter ownhome borninstate, mlabel(stateab)
```



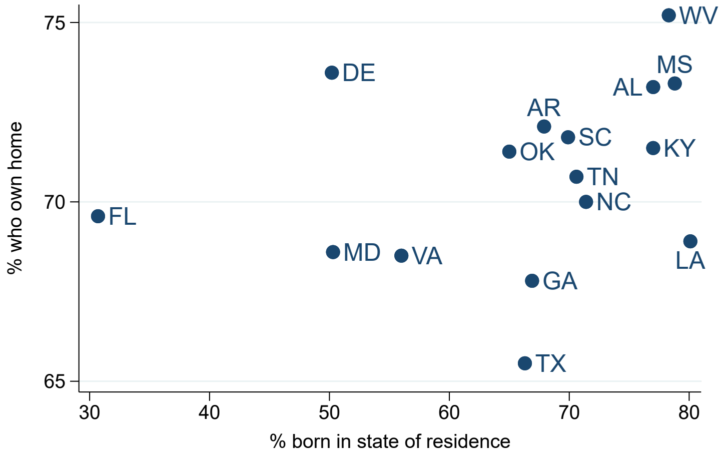
Consider this scatterplot showing the relationship between the percentage of people in a state who own their home and the percentage of people born in their state of residence. We might want to be able to identify some of the observations, and we can use the `mlabel()` (marker label) option to label the markers with the two-letter abbreviation of the state. *Uses allstates3.dta*

```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabpos(12)
```

We can use the `mlabpos()` (marker label position) option to give the marker labels a different position. Here we place the marker labels in the twelve o'clock position (above the markers). *Uses allstates3.dta*



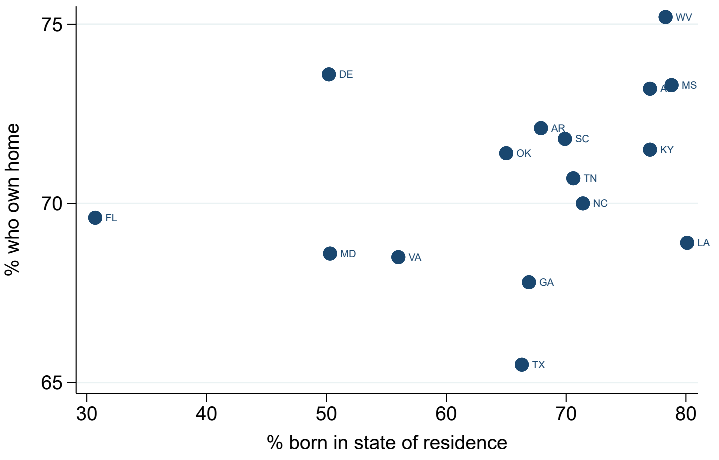
```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabvpos(pos)
```



The `mlabvpos()` (marker label variable position) option allows us to assign a different marker label position for each observation using a variable in the data file. The variable `pos` has a value of 3, placing the markers in the three o'clock position, except for states AL, MS, and LA, where `pos` is 9, 12, and 6, respectively. *Uses allstates3.dta*

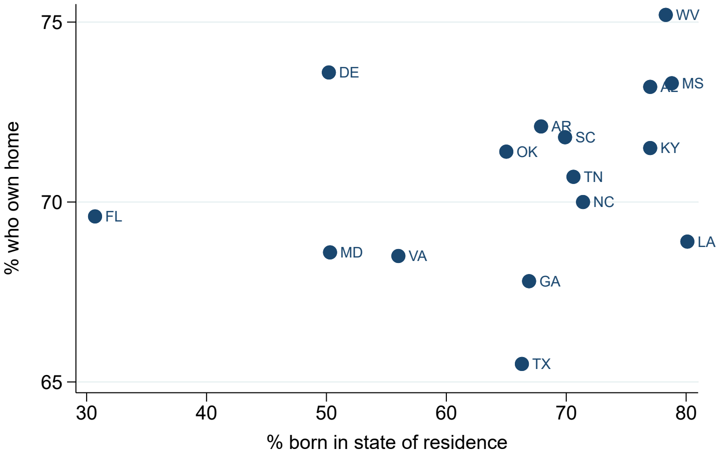
```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabsize(vsmall)
```

The `mlabsize()` (marker label size) option controls the size of the marker labels. Here we make the marker labels very small. Some other sizes you could choose include `small`, `medsmall`, `medium`, `medlarge`, `large`, and `vlarge`; see **Styles : Textsize** (section 10.12) for more details (including sizing using points, inches, and centimeters). *Uses allstates3.dta*



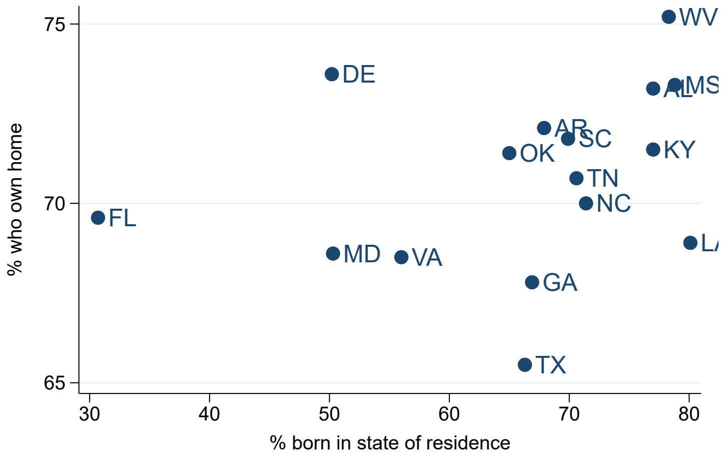
```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabsize(*.6)
```

We can also specify the `mlabsize()` option as a multiple of the original size. Here the labels are 0.6 times their normal size; see **Styles : Textsize** (section [10.12](#)) for more details (including sizing using points, inches, and centimeters). *Uses allstates3.dta*

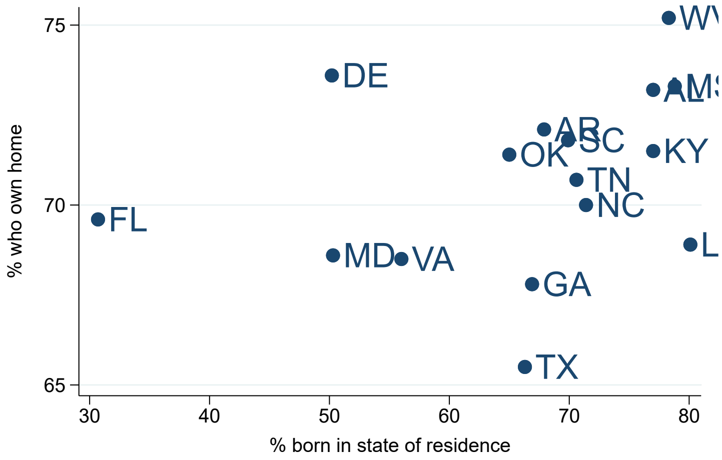


```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabsize(5rs)
```

This example illustrates another way we can control the marker label size, using `mlabsize(5rs)`. This specifies that the marker label will be 5% of the size of the graph (that is, the smaller of the width and height). This graph is 3 by 2 inches, so this works out to 5% of 2, or 0.1 inch. See **Styles : Textsize** (section [10.12](#)) for more details (including sizing using points, inches, and centimeters). *Uses allstates3.dta*



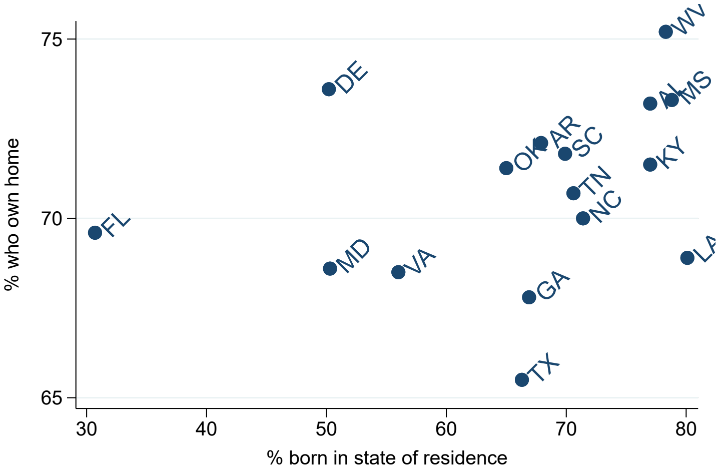
```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabsize(10pt)
```



You can also specify the marker label size using points, inches, or centimeters. This example specifies the marker labels as 10 points. See Styles : Textsize (section 10.12) for more details (including sizing using points, inches, and centimeters). Also see section 9.3.1 regarding resizing graphs that use absolute units. *Uses allstates3.dta*

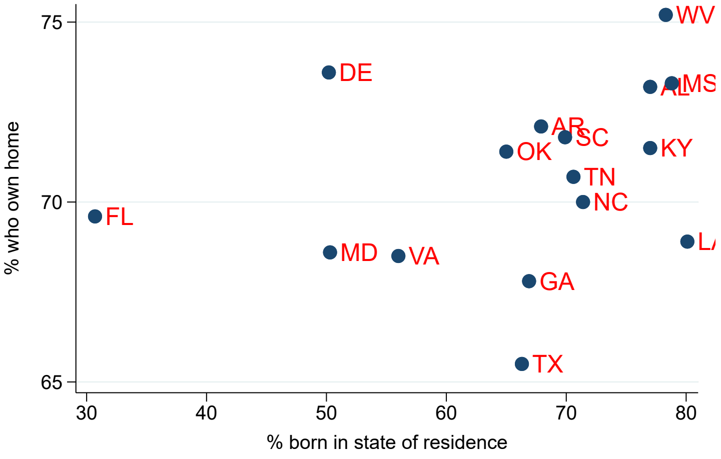
```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabangle(45)
```

The `mlabangle()` (marker label angle) option controls the angle of the marker label. For example, 0 degrees indicates horizontal text, 90 degrees vertical text, 180 degrees reverse horizontal text, and 270 degrees reverse vertical text. We can also specify negative degrees (for example, -90 degrees is the same as 270 degrees). See Styles : Angles (section 10.1) for more details. *Uses allstates3.dta*



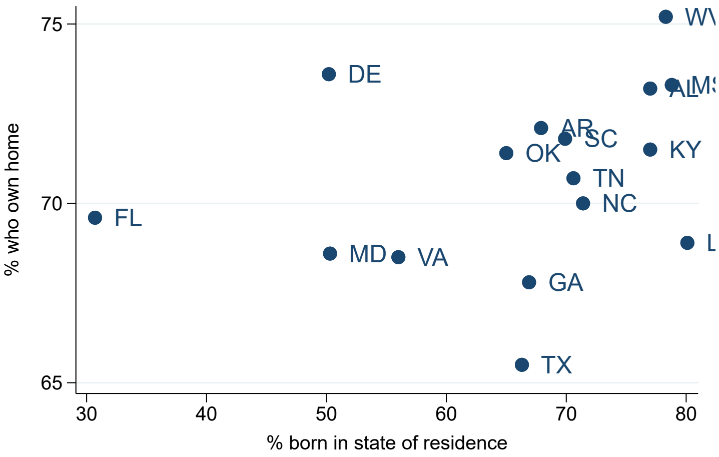
```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabcolor(red)
```

The `mlabcolor()` (marker label color) option controls the color of the marker labels. Here we make the marker labels red. See Styles : Colors (section 10.2) for more details. *Uses allstates3.dta*



```
twoway scatter ownhome borninstate, mlabel(stateab)
mlabgap(*3)
```

The `mlabgap()` (marker label gap) option controls the gap between the marker and the marker label. Here we make the gap three times the size that it would normally be. We can also specify a value less than 1 to place the marker label closer to the marker. *Uses allstates3.dta*

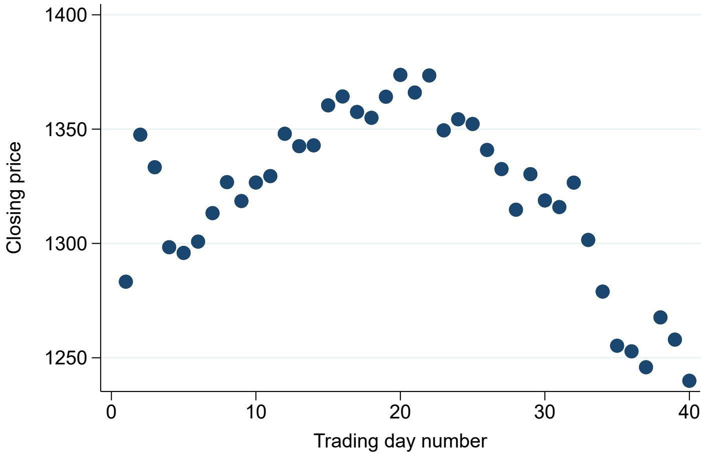


% who own home
75
70
65
% born in state of residence
30
40
50
60
70
80
WV
DE
MS
AL
AR
SC
OK
KY
TN
NC
FL
L
MD
VA
GA
TX

8.3 Connecting points and markers

Stata supports a variety of methods for connecting points by using different values for the *connectstyle*. These include `l` to connect with a straight line, `L` to connect with a straight line only if the current x value is greater than the prior x value, `J` for stairstep, `stepstair` for step then stair, and `i` for invisible connections. For the next few examples, we will use the `spjanfeb2001` data file, keeping just the data for January and February of 2001. These examples of connect styles do not demonstrate how you would normally use these styles but illustrate the different ways we can connect points. See `help connectstyle` for more information.

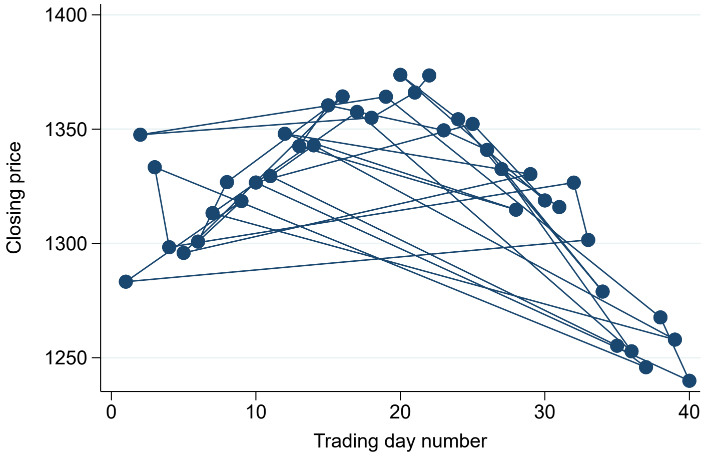
```
twoway scatter close tradeday
```



Consider this graph showing the closing price of the S&P 500 index for January and February of 2001 by `tradeday`, the trading day numbered from 1 to 40. The graph would be easier to follow if the points were connected. *Uses spjanfeb2001.dta*

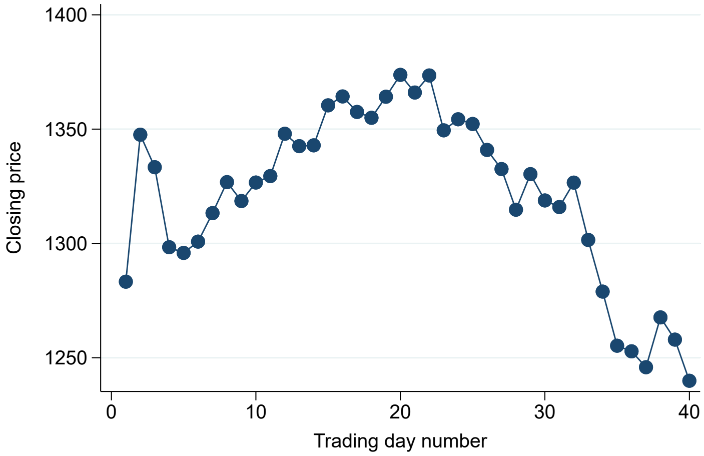
```
twoway connected close tradeday
```

We can change `scatter` to `connected` to connect the points. This, however, does not lead to the kind of graph we really wanted to create, because the observations are connected in the order in which they appear in the data file. We would obtain the desired graph if the observations were sorted according to `tradeday` or if we used the `sort` option as shown in the next example. *Uses spjanfeb2001.dta*



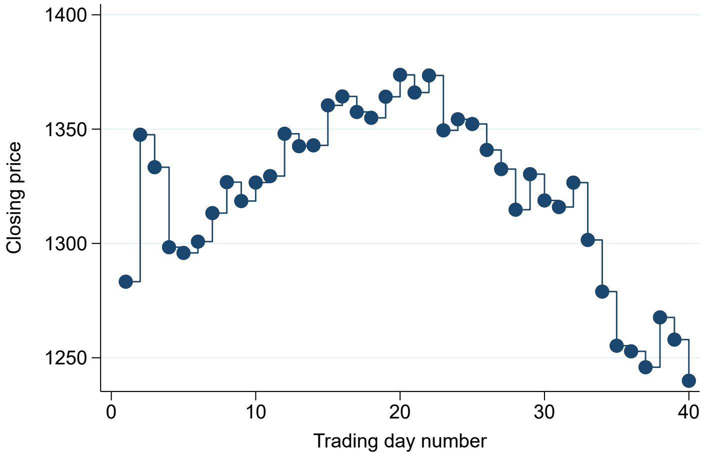
```
twoway connected close tradeday, sort
```

Here, by adding the `sort` option, the observations are connected after sorting them by `tradeday`. We could have first typed `sort tradeday`, and all subsequent graphs would have been ordered on `tradeday`, even without the `sort` option. *Uses spjanfeb2001.dta*



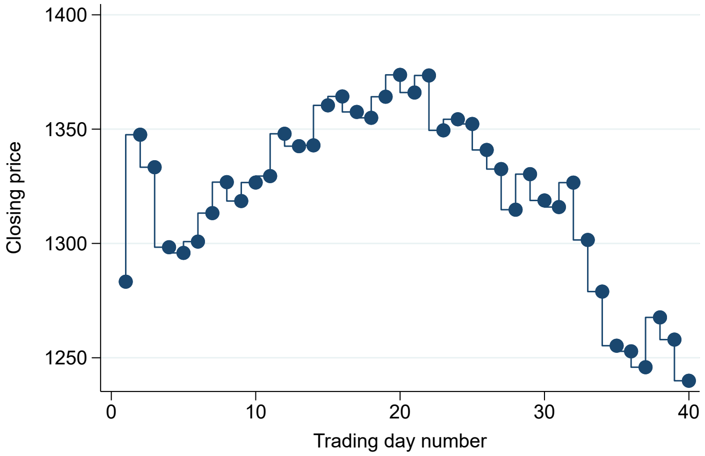
```
twoway connected close tradeday, connect(J) sort
```

This connection method, obtained by using the `connect(J)` option, would normally be used in a graph showing a survival function over time. *Uses spjanfeb2001.dta*



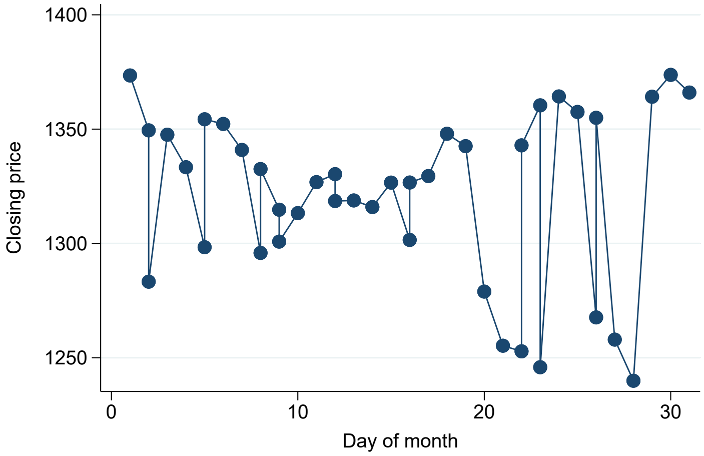
```
twoway connected close tradeday, connect(stepstair) sort
```

We can obtain a connection method related to the one above by using the `connect(stepstair)` option. *Uses spjanfeb2001.dta*



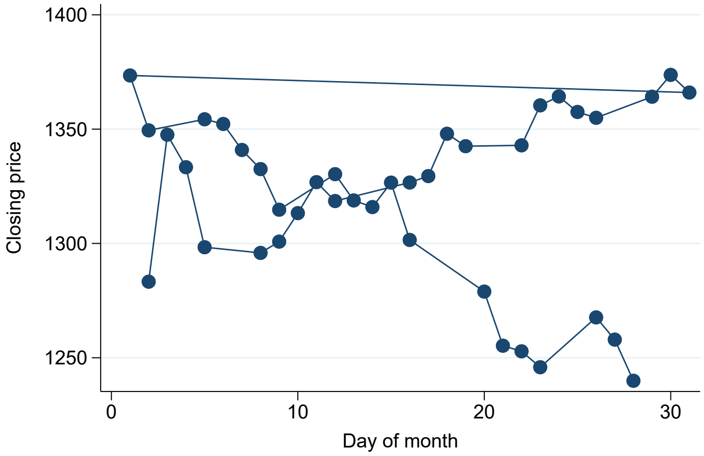
```
twoway connected close dom, sort
```

Say that we wanted to show the closing price as a function of the day of the month for the two months for which we have data. Here we have the variable `dom` (day of the month) on the x axis. With the `sort` option, the data are shown as one continuous line, as opposed to having one line for January and a second line for February. *Uses spjanfeb2001.dta*



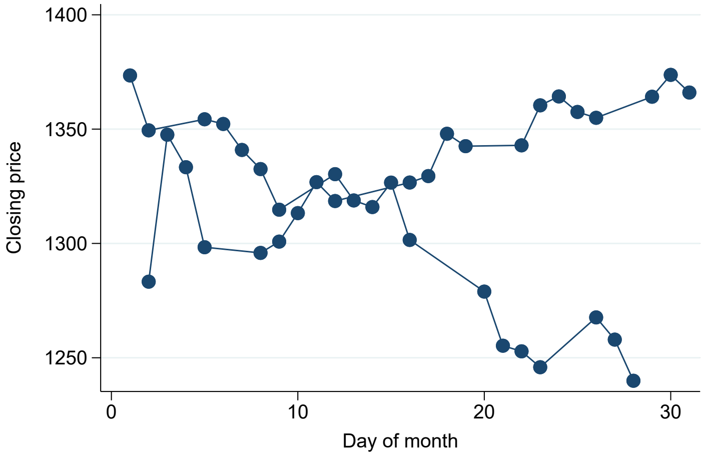
```
twoway connected close dom, sort(tradeday)
```

The `sort(tradeday)` option sorts the observations by `tradeday`. This graph is almost what we want, but the observation for January 31 is connected to the observation for February 1. *Uses spjanfeb2001.dta*

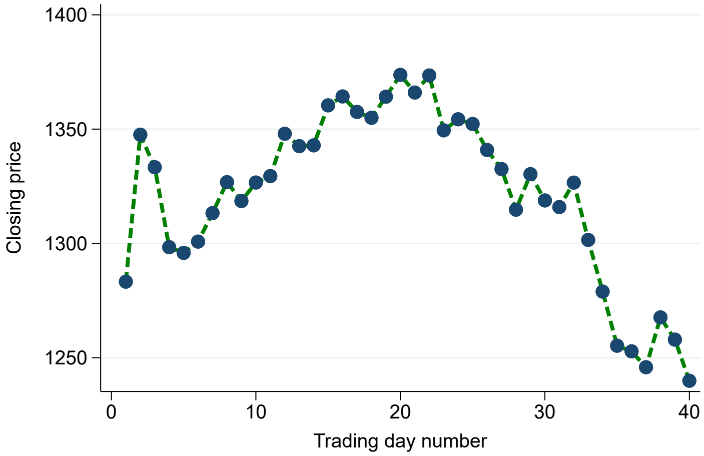


```
twoway connected close dom, connect(L) sort(tradeday)
```

This graph is what we wanted to create. The `connect(L)` option avoids the line connecting January 31 and February 1 because it connects points only as long as `dom` is increasing. When `dom` decreases from 31 to 1, the `connect(L)` option does not connect those two points. See **Styles : Connect** (section 10.5) for more details on `connect()` options. *Uses spjanfeb2001.dta*



```
twoway connected close tradeday, sort lcolor(green)
lwidth(thick) lpattern(dash)
```

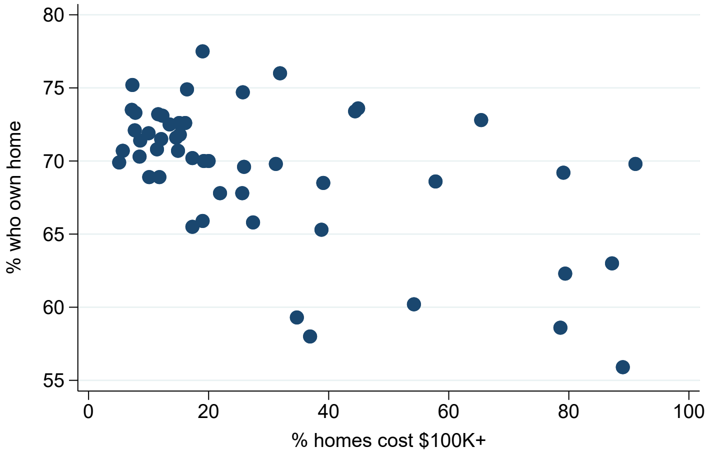


The `connect()` option determines how the markers are connected but not the color, width, or pattern of the line. Here we use the `lcolor()` (line color), `lwidth()` (line width), and `lpattern()` (line pattern) options to make the line green, thick, and dashed. See **Styles : Colors** (section 10.2), **Styles : Linewidth** (section 10.7), and **Styles : Linepatterns** (section 10.6) for more information. *Uses spjanfeb2001.dta*

8.4 Setting and controlling axis titles

This section provides more details about the use of axis-title options for providing titles for axes. For more information, see `help axis_title_options`.

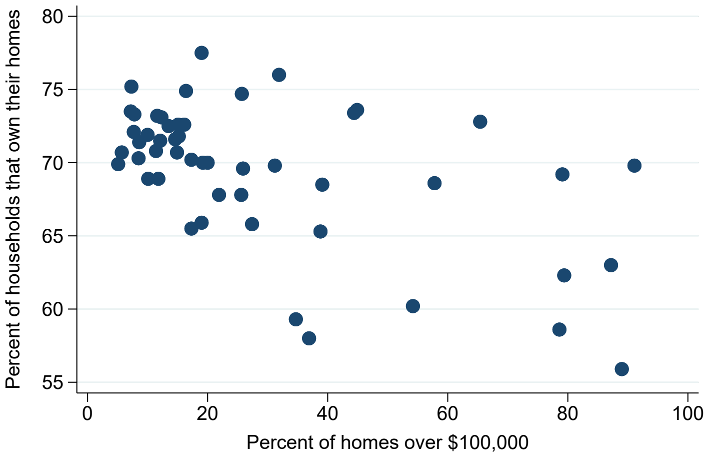
```
twoway scatter ownhome propval100
```



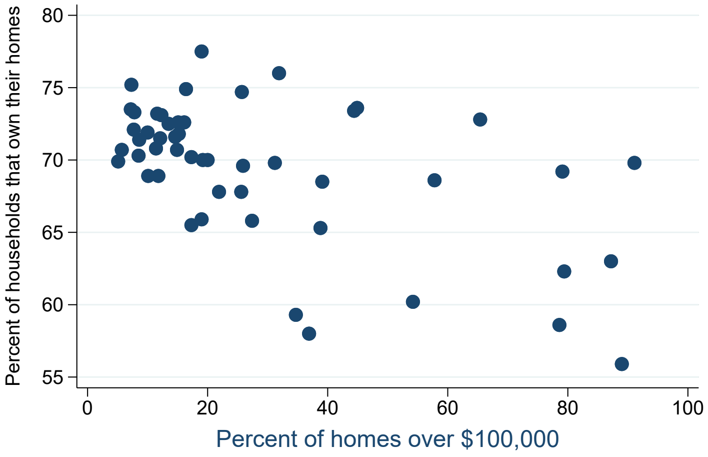
Consider this graph of the percentage of homeowners by the percentage of homes that cost over $100,000. The titles of the x and y axes are the names of the variables, unless the variables are labeled; that is, the default title is the variable label. Here the axes are labeled with the variable labels. *Uses allstatesdc.dta*

```
twoway scatter ownhome propval100, ytitle("Percent of
households that own their homes") xtitle("Percent of homes
over $100,000")
```

We can use the `ytitle()` and `xtitle()` options to supply our own titles. *Uses allstatesdc.dta*



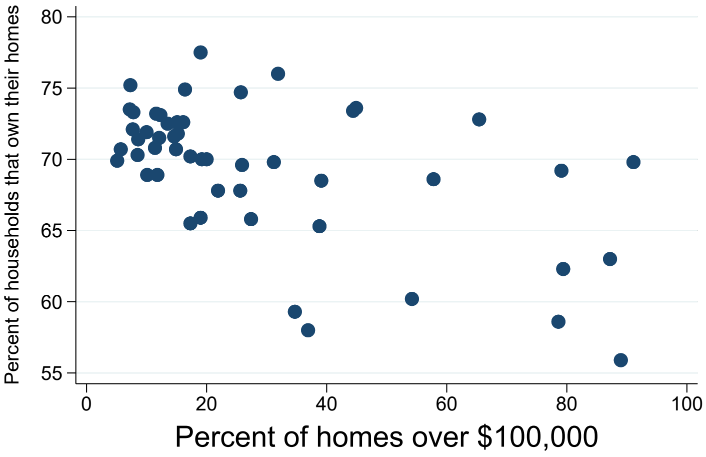
```
twoway scatter ownhome propval100, ytitle("Percent of
households that own their homes") xtitle("Percent of homes
over $100,000", size(large) color(navy))
```



Because an axis title is considered a textbox, we can use textbox options to control the look of the axis title. See Options : Textboxes (section 8.11) for more examples. Also see Styles : Textsize (section 10.12) for more on specifying text sizes (including sizing using points, inches, and centimeters). *Uses allstatesdc.dta*

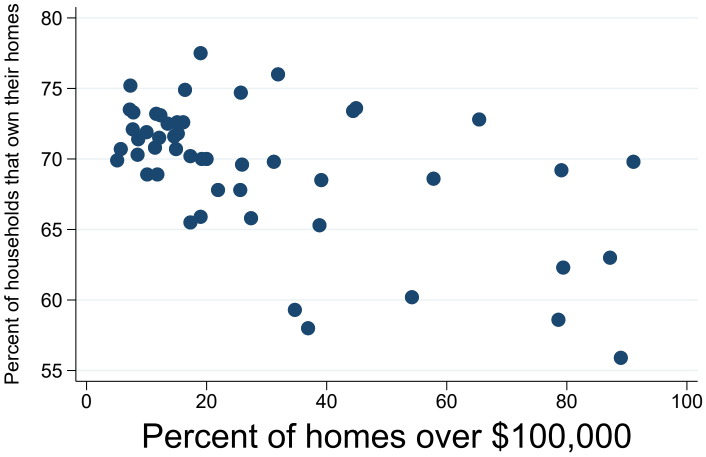
```
twoway scatter ownhome propval100, ytitle("Percent of
households that own their homes") xtitle("Percent of homes
over $100,000", size(*1.5))
```

This example uses the `size(*1.5)` option to change the size of the x-axis title, making it 1.5 times its normal size. Had we specified `size(*.7)`, the title would have been displayed to 70% of its original size. See Options : Textboxes (section 8.11) for more details on specifying the size of text (including sizing using points, inches, and centimeters). *Uses allstatesdc.dta*



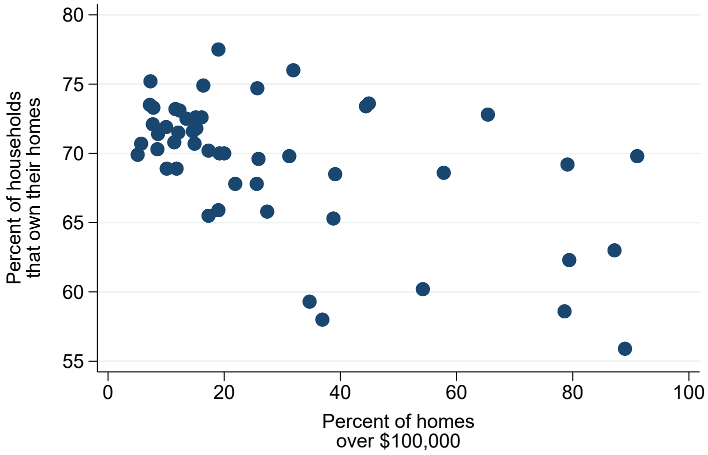
```
twoway scatter ownhome propval100, ytitle("Percent of
households that own their homes") xtitle("Percent of homes
over $100,000", size(10pt))
```

You can also specify the size of text using points, inches, or centimeters. This example specifies the size of the x-axis title as 10 points. See Options : Textboxes (section 8.11) for more details on sizing text (including sizing using points, inches, and centimeters). Also see section 9.3.1 regarding resizing graphs that use absolute units. *Uses allstatesdc.dta*



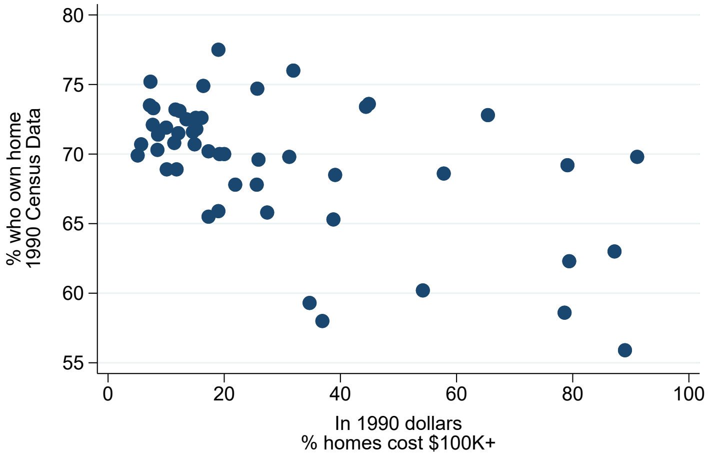
```
twoway scatter ownhome propval100, ytitle("Percent of
households" "that own their homes") xtitle("Percent of homes"
"over $100,000")
```

Here we supply the same titles but divide them into two separate, quoted strings, which are then displayed on separate lines. *Uses allstatesdc.dta*



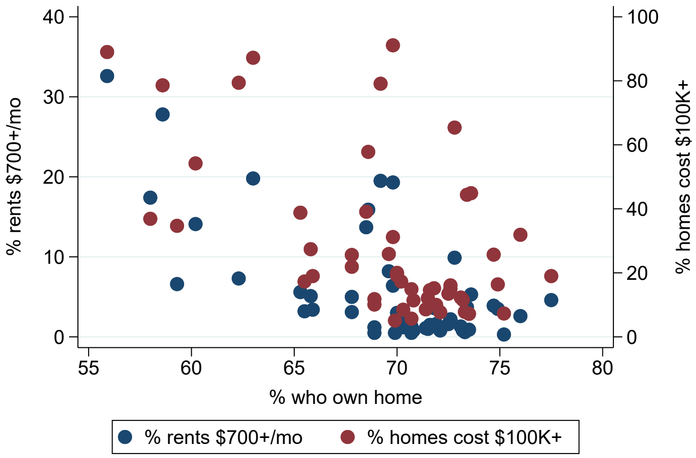
```
twoway scatter ownhome propval100, ytitle("1990 Census Data",
suffix) xtitle("In 1990 dollars", prefix)
```

We can use the `prefix` and `suffix` options to add information before or after the existing title, respectively. *Uses allstatesdc.dta*

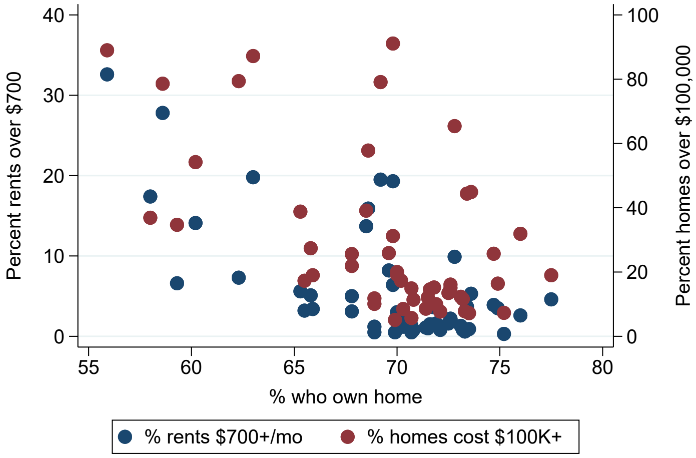


```
twoway (scatter rent700 ownhome ) (scatter propval100
ownhome, yaxis(2))
```

Consider this overlaid twoway graph. The two y variables are both scaled in percentages, but they have different ranges. We use the `yaxis(2)` option on the second `scatter` command to place that axis on the second y axis, which is then placed on the right axis. *Uses allstatesdc.dta*



```
twoway (scatter rent700 ownhome) (scatter propval100 ownhome,
yaxis(2)), ytitle("Percent rents over $700", axis(1))
ytitle("Percent homes over $100,000", axis(2))
```



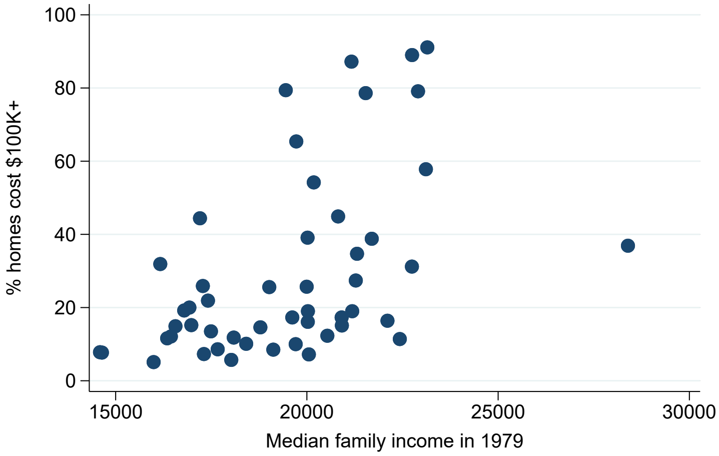
Now that we have two y axes, the `ytitle()` option would change the y title for the first y axis, unless we specify otherwise. Here we supply a `ytitle()` option with the `axis(1)` option to indicate that the title belongs to the first y axis, and a second `ytitle()` option using the `axis(2)` option to indicate that the second title belongs to the second y axis. *Uses allstatesdc.dta*

8.5 Setting and controlling axis labels

This section describes more details about axis labels, including major and minor (numeric) labels, major and minor tick marks, and grid lines. This section also shows how to control the appearance of these objects (for example, size, color, thickness, or angle). For more information, see `help axis_label_options`.

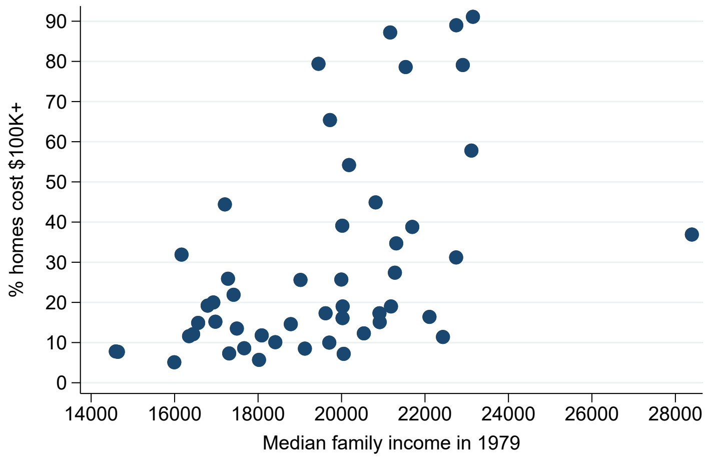
```
twoway scatter propval100 faminc
```

Let's start with a basic graph showing the percentage of homes costing over $100,000 by the median family income. *Uses allstatesdc.dta*



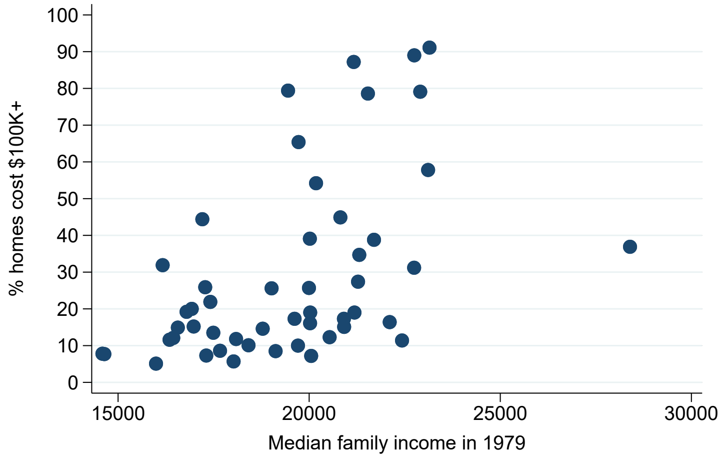
```
twoway scatter propval100 faminc, xlabel(#10) ylabel(#10)
```

By using the `xlabel(#10)` and `ylabel(#10)` options, we ask for about 10 values to be labeled on each axis. Stata chose to use 10 values for the y axis, labeling it from 0 to 90, incrementing by 10, and 8 values for the x axis going from 14,000 to 28,000, incrementing by 2,000. Sometimes Stata follows the suggestion exactly, and sometimes it chooses a different number of values to make more logical labels. *Uses allstatesdc.dta*



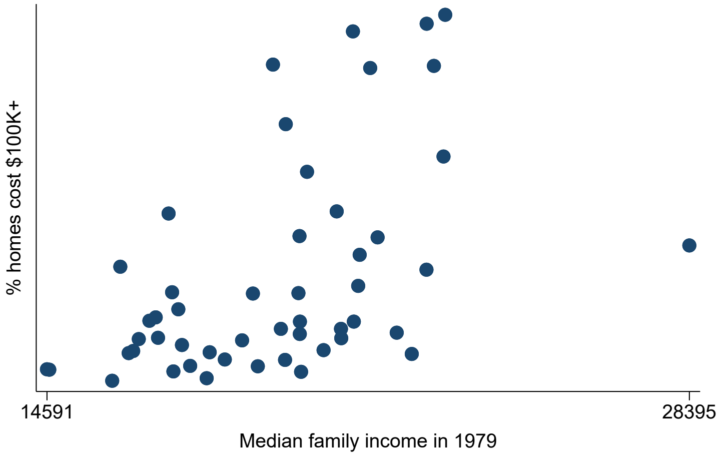
```
twoway scatter propval100 faminc, ylabel(0(10)100)
```

We can change the major labels for the y variable to range from 0 to 100, incrementing by 10, by using the `ylabel(0(10)100)` option. *Uses allstatesdc.dta*



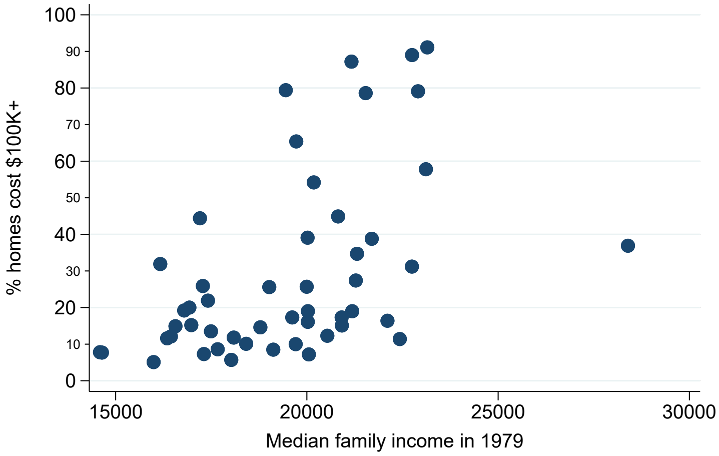
```
twoway scatter propval100 faminc, xlabel(minmax) ylabel(none)
```

Here we use the `xlabel(minmax)` option to label the x axis with only the minimum and maximum and use `ylabel(none)` so that the y axis will have no major labels or ticks. *Uses allstatesdc.dta*



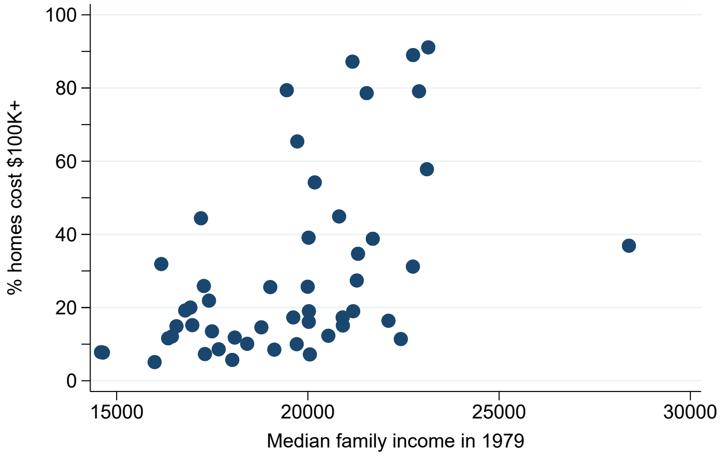
```
twoway scatter propval100 faminc, ymlabel(10(20)90)
```

The default graph had major labels for the y axis at 0, 20, 40, 60, 80, and 100. We add minor labels for the y variable at 10, 30, 50, 70, and 90 by using the `ymlabel(10(20)90)` option. The `m` in `ymlabel()` stands for minor. *Uses allstatesdc.dta*



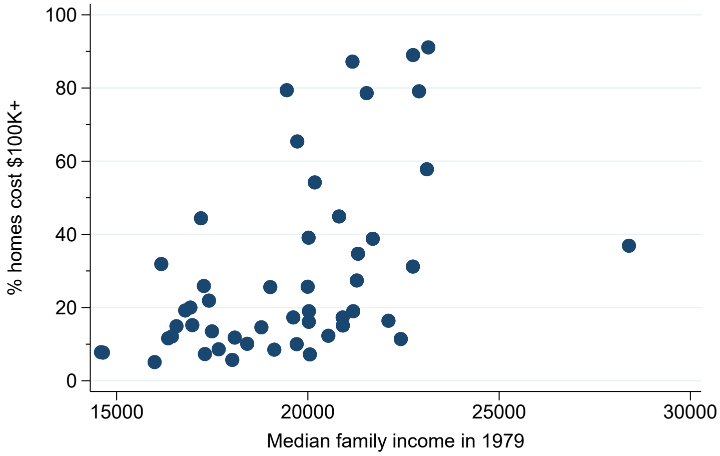
```
twoway scatter propval100 faminc, ytick(10(10)90)
```

By using the `ytick(10(10)90)` option, we change the major ticks to range from 10 to 90, incrementing by 10. *Uses allstatesdc.dta*



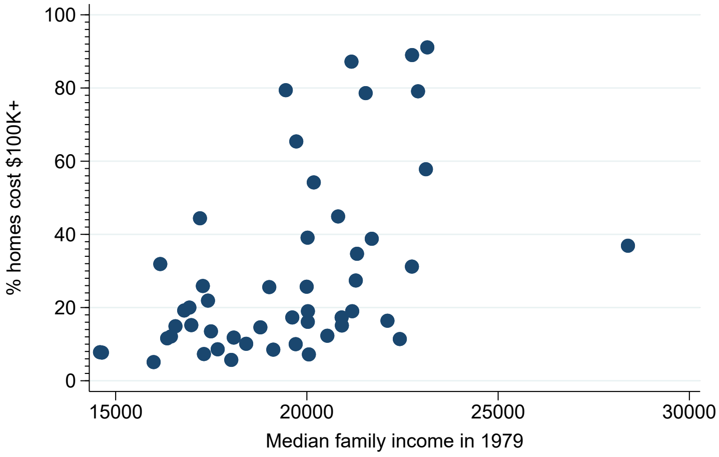
```
twoway scatter propval100 faminc, ymtick(10(20)90)
```

The `ymtick()` option adds minor ticks to the graph. Here we add minor ticks at 10, 30, 50, 70, and 90. The `m` in `ymtick()` stands for minor. *Uses allstatesdc.dta*



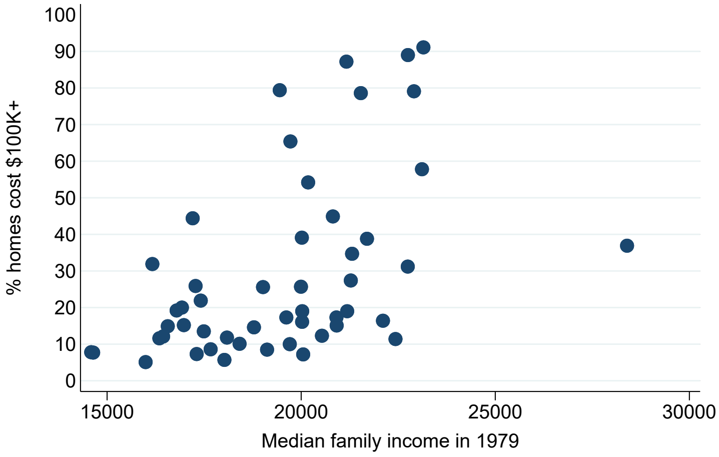
```
twoway scatter propval100 faminc, ymtick(##10)
```

The default graph had major labels for the y axis at 0, 20, 40, 60, 80, and 100. We can place 9 minor ticks between the major ticks with the `ymtick(##10)` option. The value of 10 includes the 9 minor ticks plus the 10th major tick. *Uses allstatesdc.dta*



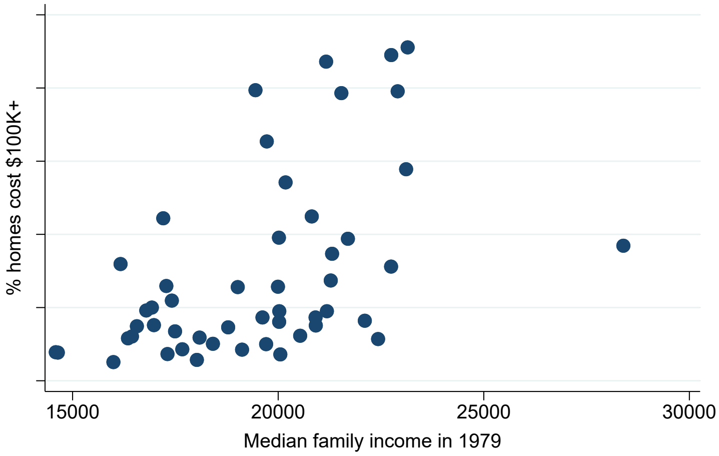
```
twoway scatter propval100 faminc, ylabel(0(10)100, noticks)
```

We specify the `noticks` option to label the y axis using values ranging from 0 to 100, incrementing by 10, but suppressing the display of ticks. *Uses allstatesdc.dta*



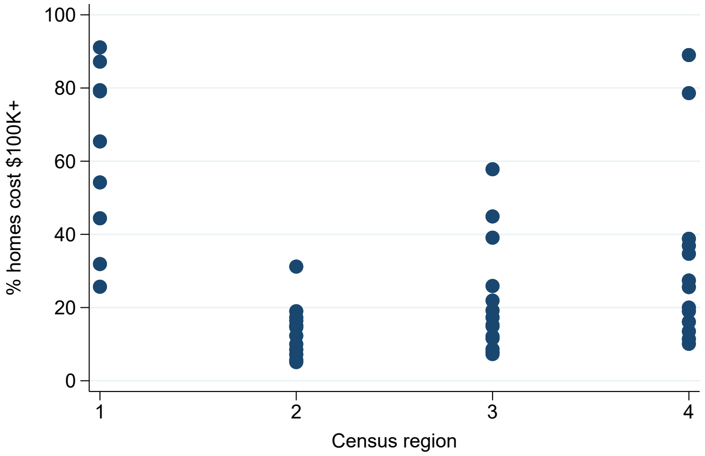
```
twoway scatter propval100 faminc, ylabel(, nolabel)
```

We suppress the labels by using the `nolabel` option, thus showing only the ticks. *Uses allstatesdc.dta*



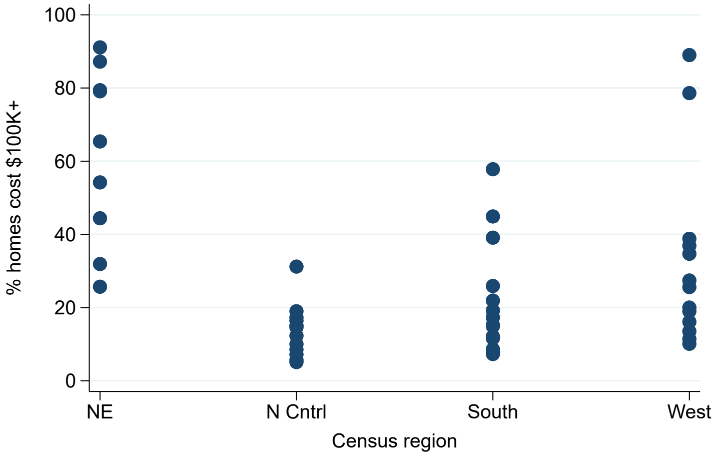
```
twoway scatter propval100 region
```

If a variable has meaningful value labels, we can display the value labels instead of the values. For example, we can look at `propval100` broken down by census region, but we do not know which regions correspond to the values 1–4. *Uses allstatesdc.dta*



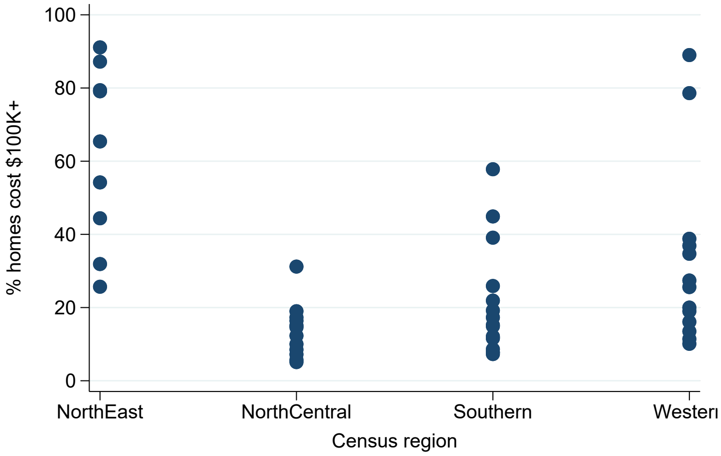
```
twoway scatter propval100 region, xlabel(, valuelabels)
```

With the `xlabel(, valuelabels)` option, the value labels are displayed, making the graph much easier to understand. *Uses allstatesdc.dta*



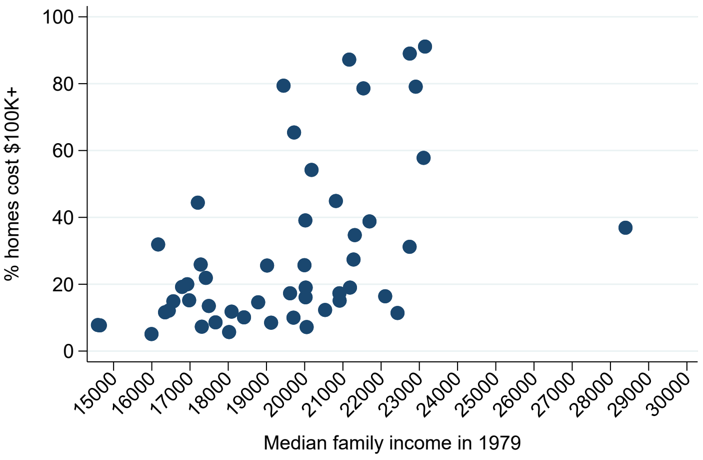
```
twoway scatter propval100 region, xlabel(1 "NorthEast" 2
"NorthCentral" 3 "Southern" 4 "Western")
```

We can use the `xlabel()` option to add labels to the x axis. *Uses allstatesdc.dta*



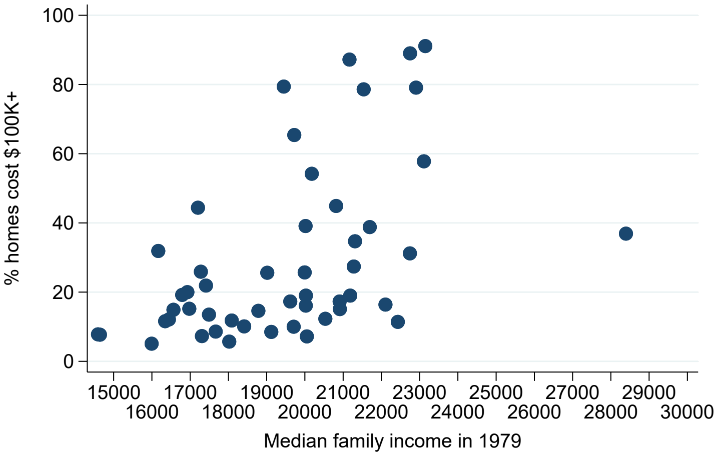
```
twoway scatter propval100 faminc, xlabel(15000(1000)30000,
angle(45))
```

If we label an axis with a large number of values (and especially with wide values), the labels may crowd each other and overlap. Here we label the x axis from 15000 to 30000, incrementing by 1000. To avoid overlapping, we add the `angle(45)` option to show the labels at a 45-degree angle. *Uses allstatesdc.dta*



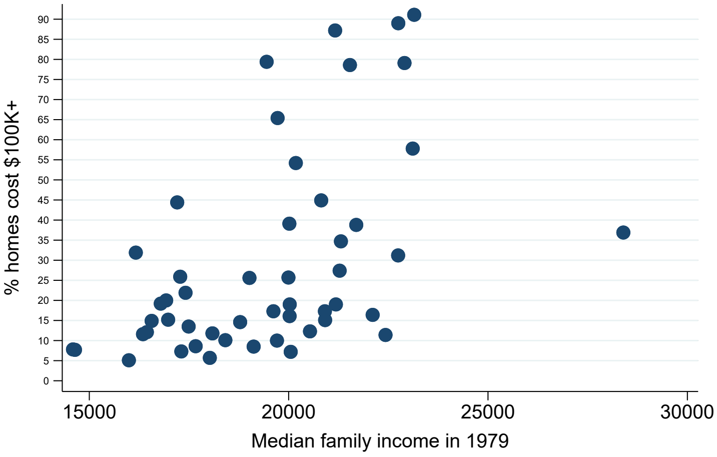
```
twoway scatter propval100 faminc, xlabel(15000(1000)30000,
alternate)
```

We can also avoid overlapping the axis labels by adding the `alternate` option to `xlabel()`. The labels are now displayed in two rows of alternating text, so they are not crowded or overlapped. *Uses allstatesdc.dta*



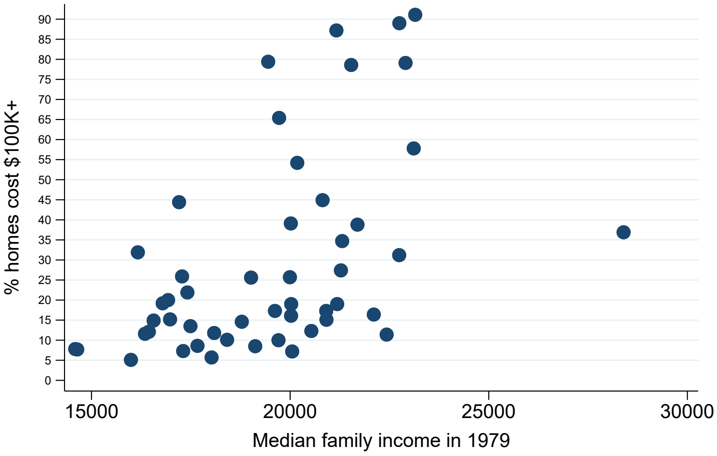
```
twoway scatter propval100 faminc, ylabel(0(5)90,
labsize(vsmall))
```

We can control the size of labels with the `labsize()` option. Here we label our y axis from 0 to 90, incrementing by 5, but we specify `labsize(vsmall)` so that smaller labels are used to minimize crowding and overlapping labels. See **Styles : Textsize** (section [10.12](#)) for more information on controlling the size of text (including sizing using points, inches, and centimeters). *Uses allstatesdc.dta*



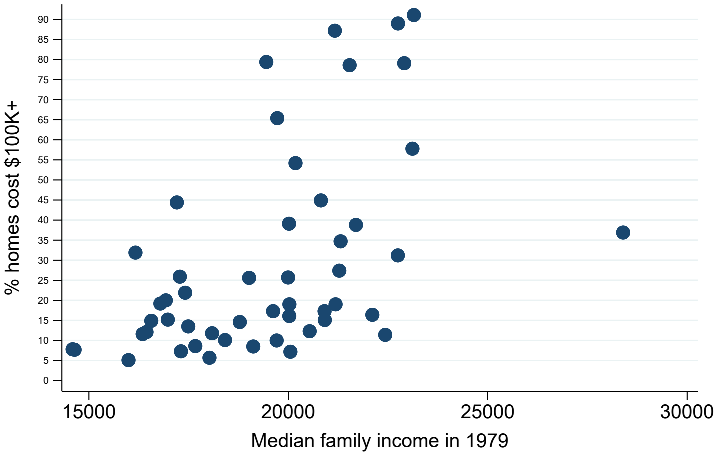
```
twoway scatter propval100 faminc, ylabel(0(5)90,
labsize(*0.6))
```

We can also control the size of labels by specifying a multiplier of the original label size. This example reduces the labels to 60% of their original size. Had we specified, for example, `labsize(*1.4)`, the labels would have been shown as 1.4 times their original size. See **Styles : Textsize** (section [10.12](#)) for more information on controlling the size of text (including sizing using points, inches, and centimeters). *Uses allstatesdc.dta*



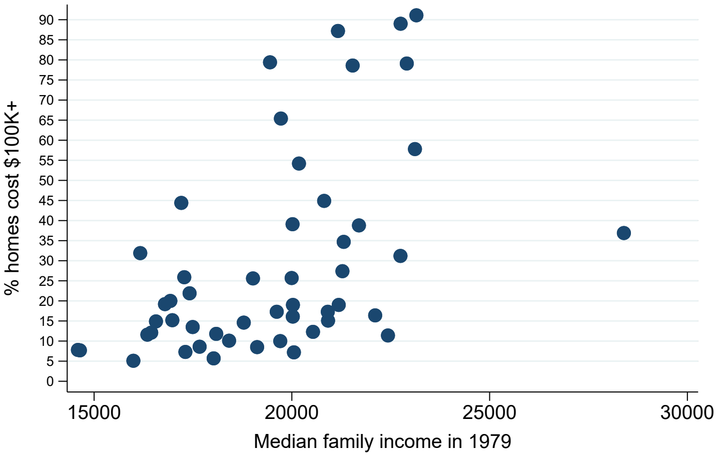
```
twoway scatter propval100 faminc, ylabel(0(5)90,
labsize(2rs))
```

In this example, we specify `labsize(2rs)` to size the y axis as 2% of the size of the graph (the minimum of the width and height). The graph is 2 by 3 inches, so the `labsize(2rs)` option resolves to 2% of 2 inches, 0.04 inch. See Styles : Textsize (section 10.12) for more information on controlling the size of text (including sizing using points, inches, and centimeters). *Uses allstatesdc.dta*



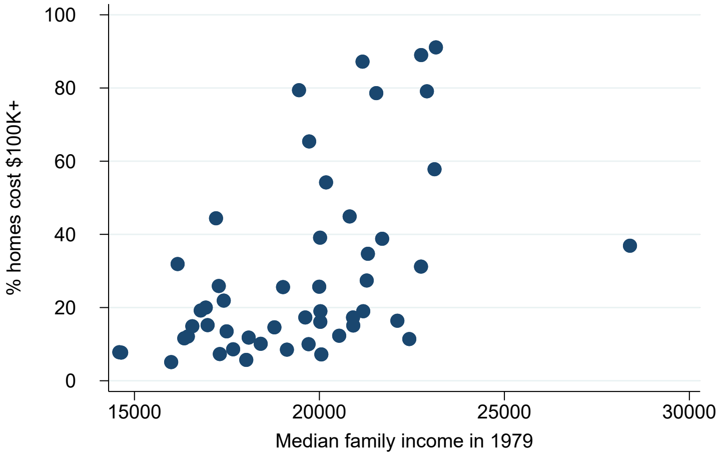
```
twoway scatter propval100 faminc, ylabel(0(5)90,
labsize(4pt))
```

We can also specify the size of the label using points, inches, or centimeters. By specifying `labsize(4pt)`, we display the size of the labels as 4 points. See Styles : Textsize (section 10.12) for more information on controlling the size of text (including sizing using points, inches, and centimeters). Also see section 9.3.1 regarding resizing graphs that use absolute units. *Uses allstatesdc.dta*



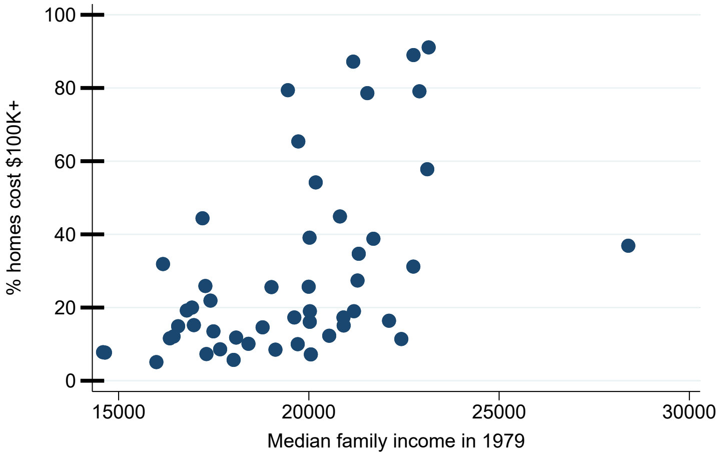
```
twoway scatter propval100 faminc, ylabel(, labgap(large))
```

We can control the gap between the label and the tick with the `labgap()` option. In this example, we make the gap between the y labels and the y ticks `large`. *Uses allstatesdc.dta*



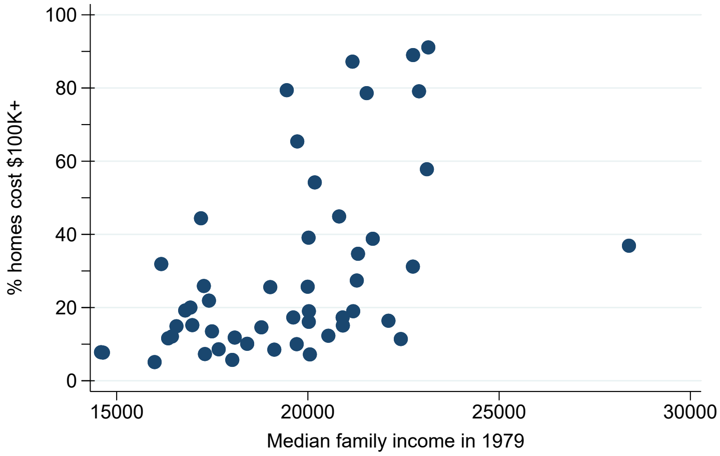
```
twoway scatter propval100 faminc, ylabel(, tlength(large)
tlwidth(thick) tposition(crossing))
```

Here we use the `tlength()`, `tlwidth()`, and `tposition()` options to make the tick length large, the width thick, and the ticks cross the y axis. *Uses allstatesdc.dta*



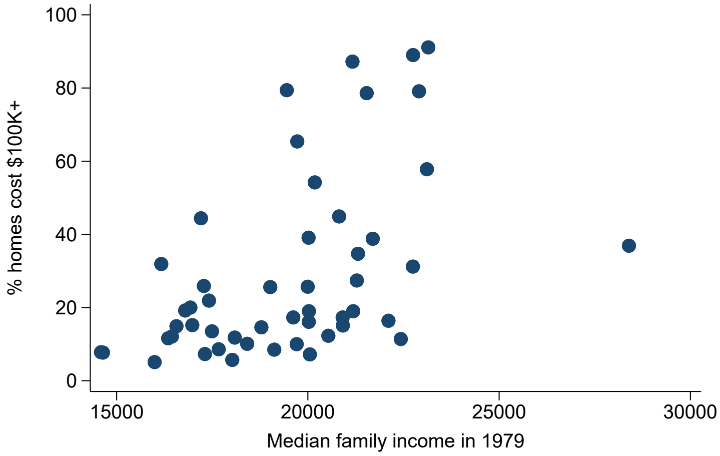
```
twoway scatter propval100 faminc, ytick(#10,
tposition(outside)) ymtick(#5, tposition(inside))
```

In this example, we suggest 10 major ticks and 5 minor ticks, locating the major ticks on the outside of the plot and the minor ticks on the inside of the plot. *Uses allstatesdc.dta*



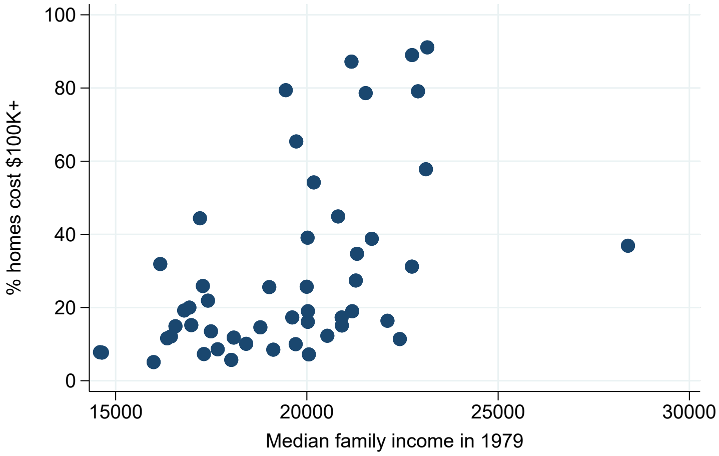
```
twoway scatter propval100 faminc, ylabel(, nogrid)
```

Here we suppress the grid corresponding to the y label by using the `nogrid` option. You can also use the `grid` and `nogrid` options with the `ymlabel()`, `ytick()`, and `ymtick()` options as well as the `xlabel()`, `xmlabel()`, `xtick()`, and `xmtick()` options. *Uses allstatesdc.dta*



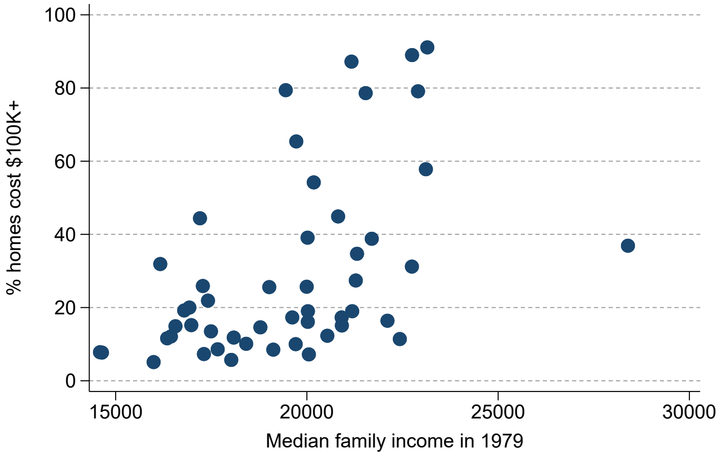
```
twoway scatter propval100 faminc, ylabel(, grid) xlabel(,
grid)
```

We display a grid for the values that correspond to the `ylabel()` and the `xlabel()` options with the `grid` option, as shown here. Depending on the scheme we choose, grids may be included or omitted by default. *Uses allstatesdc.dta*



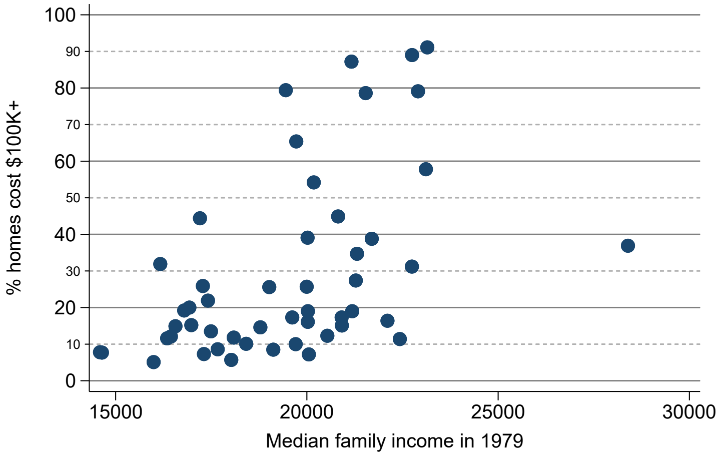
```
twoway scatter propval100 faminc, ylabel(, grid
glwidth(vthin) glcolor(gs10) glpattern(shortdash))
```

We can control the grid-line width, grid-line color, and grid-line pattern with the `glwidth()`, `glcolor()`, and `glpattern()` options. Here we make the grid line very thin, the color gray (`gs10`), and the pattern of the lines short dashes. See **Styles : Linewidth** (section 10.7), **Styles : Colors** (section 10.2), and **Styles : Linepatterns** (section 10.6) for more details. *Uses allstatesdc.dta*



```
twoway scatter propval100 faminc, ylabel(0(20)100, grid
glcolor(gs8) glpattern(solid)) ymlabel(10(20)90, grid
glcolor(gs11) glpattern(shortdash))
```

This example shows a solid, darker gray line for the major axis labels and a lighter gray, short, dashed line for the minor axis labels. *Uses allstatesdc.dta*

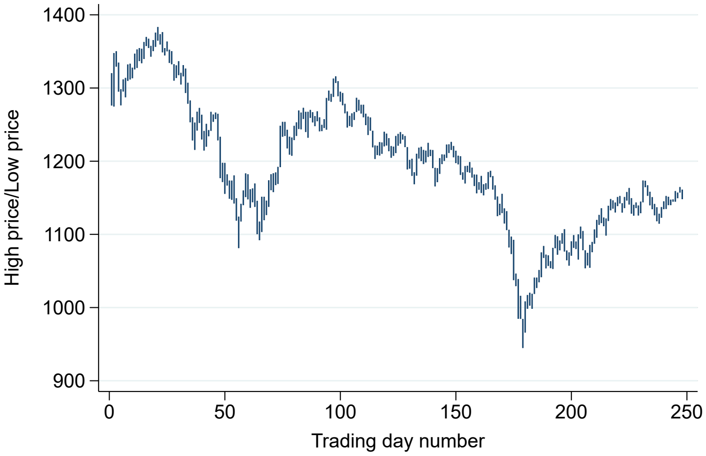


8.6 Controlling axis scales

This section provides more details about axis-scale options, which allow us to control whether an axis is displayed, where it is displayed, the direction it is displayed, and the scale of the axis. For more information about these options, see `help axis_scale_options`. This section begins by using data on the S&P 500 from January 2, 2001, to December 31, 2001, stored in the file `sp2001`. For simplicity, we will use `tradeday` on the x axis, representing the trading day of the year.

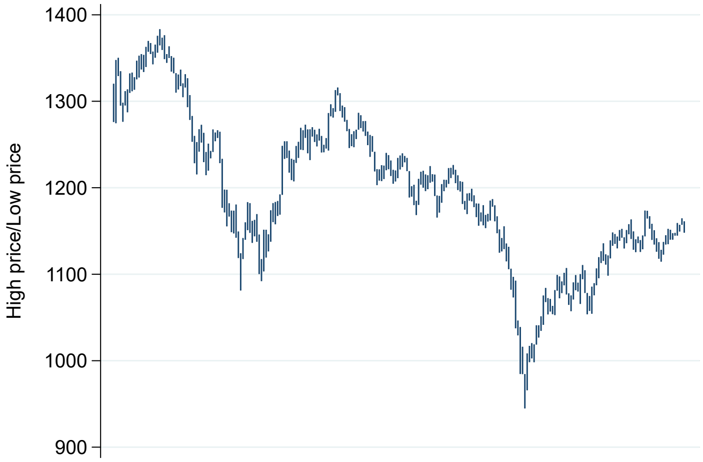
```
twoway rspike high low tradeday
```

Consider this `rspike` graph showing the high and low prices across 248 trading days. *Uses sp2001.dta*



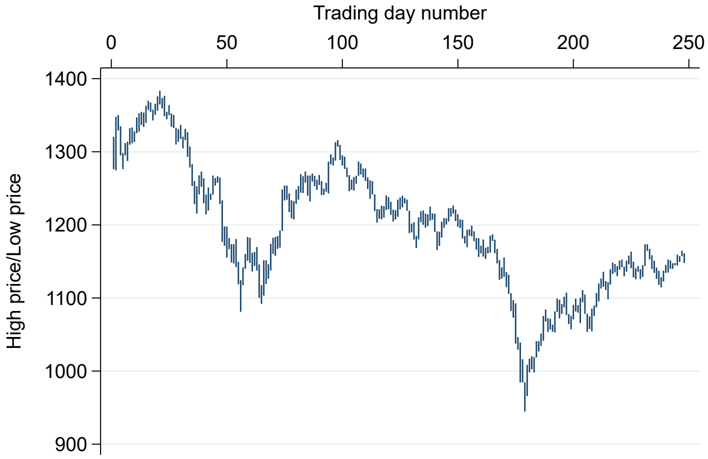
```
twoway rspike high low tradeday, xscale(off)
```

The `xscale(off)` option removes the display of the x axis entirely. Although it is not shown, you could do the same for the y axis by using the `yscale(off)` option. This option can be useful when combining multiple graphs on the same scale. *Uses sp2001.dta*



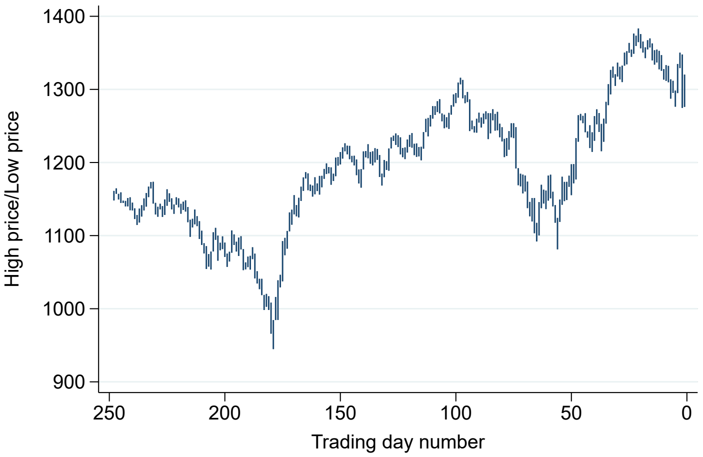
```
twoway rspike high low tradeday, xscale(alt)
```

We shift the display of the x axis from the bottom of the graph to the top of the graph by using the `xscale(alt)` option. Likewise, you can shift the y axis from the left to the right by using the `yscale(alt)` option. *Uses sp2001.dta*



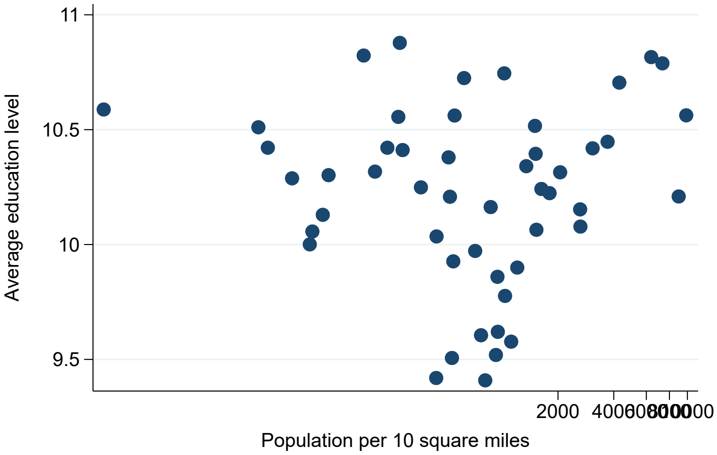
```
twoway rspike high low tradeday, xscale(reverse)
```

We reverse the scale of the x axis by specifying the `xscale(reverse)` option, as illustrated here. You can reverse the y axis by indicating the `yscale(reverse)` option. *Uses sp2001.dta*



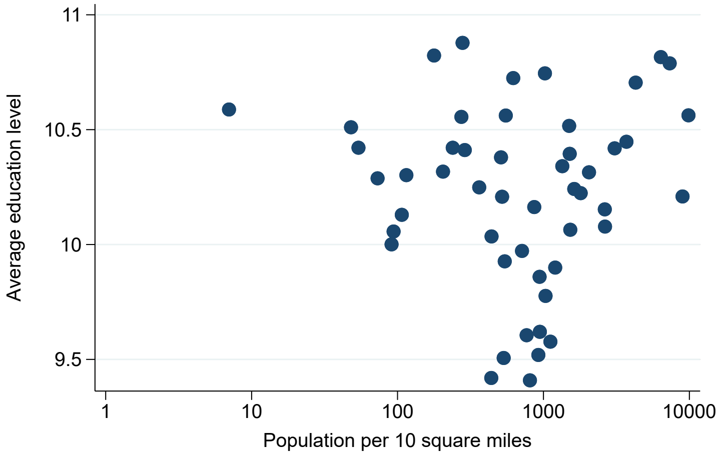
```
twoway scatter educ popden, xscale(log)
```

We briefly return to the `allstates` data file to show the `xscale(log)` option, which indicates that the x axis should be displayed on a log scale. The labels for the x axis overlap each other. *Uses allstates.dta*

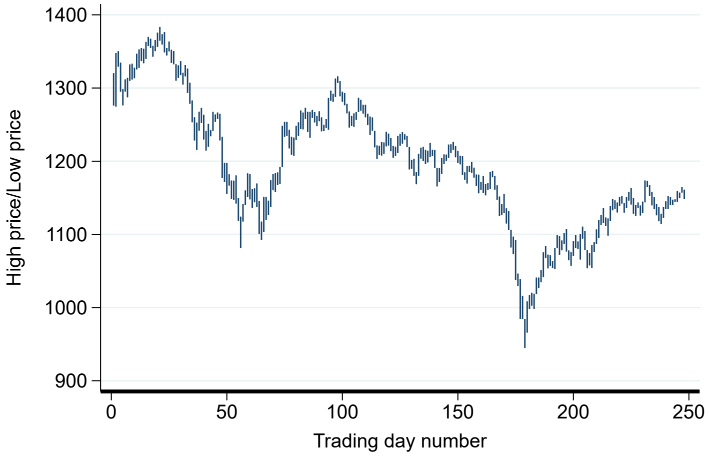


```
twoway scatter educ popden, xscale(log) xlabel(1 10 100 1000
10000)
```

Here we use the `xlabel()` option to change the labels for the x axis using the values 1, 10, 100, 1,000, and 10,000. Note how these powers of 10 are more equally spaced, reflecting the log scale of the x axis. *Uses allstates.dta*



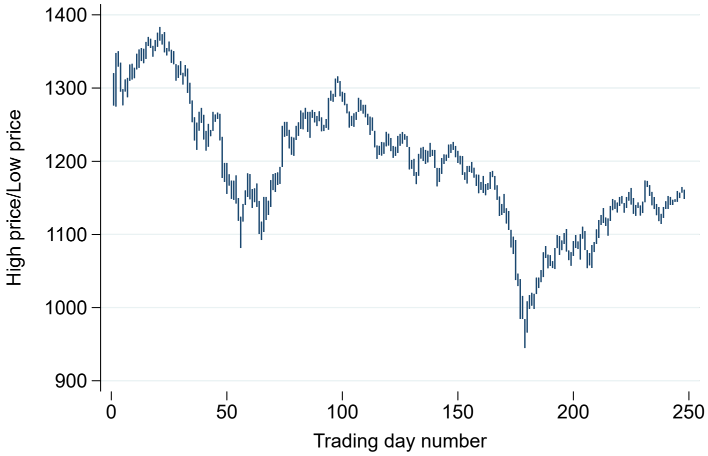
```
twoway rspike high low tradeday, xscale(lwidth(thick))
```



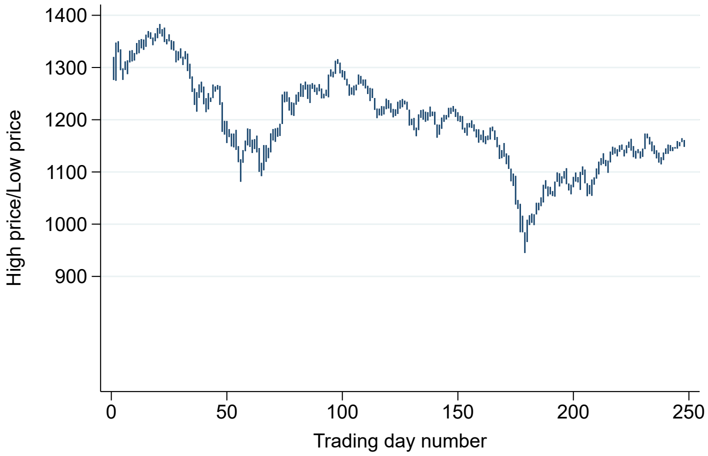
We now return to the `sp2001` data. You can use the `xscale()` and `yscale()` options to control the axis lines. Here we make the x-axis line thick by specifying `xscale(lwidth(thick))`. See Styles : Linewidth (section 10.7) for more options you can use to control line widths. *Uses sp2001.dta*

```
twoway rspike high low tradeday, xscale(noline)
```

We suppress the display of the x-axis line by using the `xscale(noline)` option. *Uses sp2001.dta*



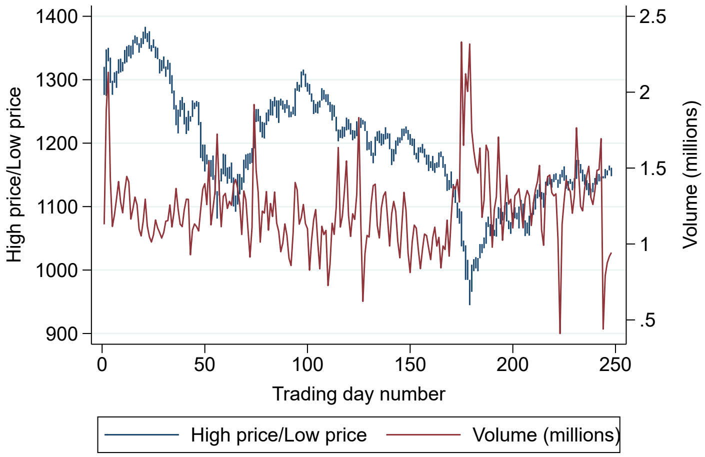
```
twoway rspike high low tradeday, yscale(range(700 1400))
```



We can use the `yscale(range())` option to expand the scale of the y axis without needing to expand the labels for the axis (as the `ylabel()` option would). Here we have expanded the y axis to range from 700 to 1400. The next examples illustrate the usefulness of this option. *Uses sp2001.dta*

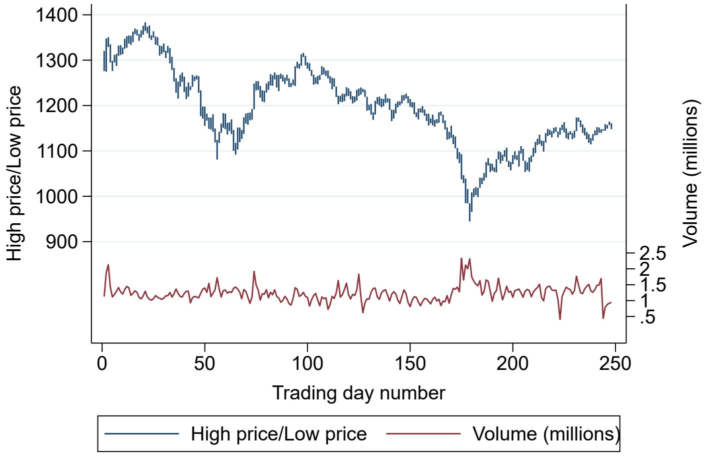
```
twoway (rspike high low tradeday) (line volmil tradeday, sort
yaxis(2))
```

Consider that, in addition to the spike graph that shows the high and low values for a given trading day, we wish to see the volume for a given trading day. We can combine the plots into one graph, but this is difficult to read because the two plots overlap. *Uses sp2001.dta*



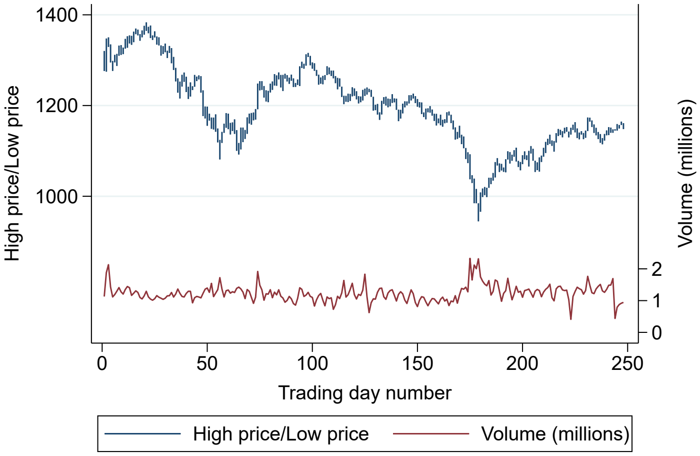
```
twoway (rspike high low tradeday) (line volmil tradeday, sort
yaxis(2)), yscale(range(700 1400) axis(1)) yscale(range(0 10)
axis(2))
```

The `yscale(range(700 1400) axis(1))` option sets the range of the left y axis to occupy the upper third of the graph, whereas the `yscale(range(0 10) axis(2))` option sets the range of the right y axis to occupy the lower third of the graph. *Uses sp2001.dta*



```
twoway (rspike high low tradeday) (line volmil tradeday, sort
yaxis(2)), yscale(range(700 1400) axis(1)) yscale(range(0 10)
axis(2)) ylabel(1000 1200 1400, axis(1)) ylabel(0 1 2,
axis(2))
```

We add the `ylabel(1000 1200 1400, axis(1))` and `ylabel(0 1 2, axis(2))` options to the previous example to make the labels for the y axes more readable. *Uses sp2001.dta*

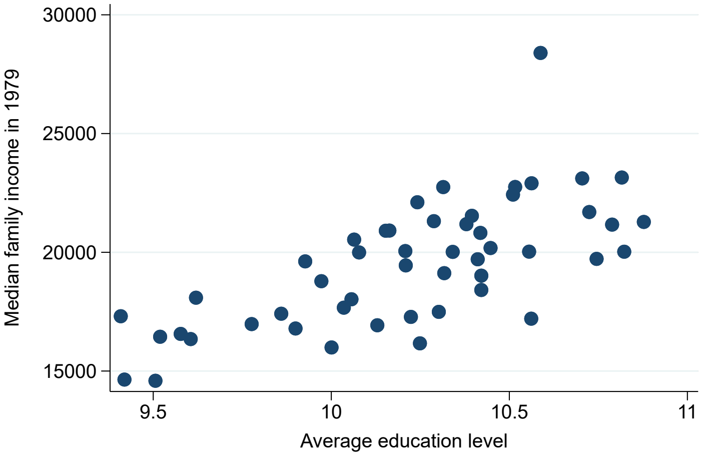

1400
1200
1000
High price/Low price
Volume (millions)
2
1
0
0
50
100
150
200
250
Trading day number
High price/Low price
Volume (millions)

8.7 Selecting an axis

This section provides more details about how to select different axes and how to modify them. By default, any modifications you make to an axis are applied to the first axis, so you need to take extra action to modify other axes that you may create. For more information about these options, see `help axis_choice_options`.

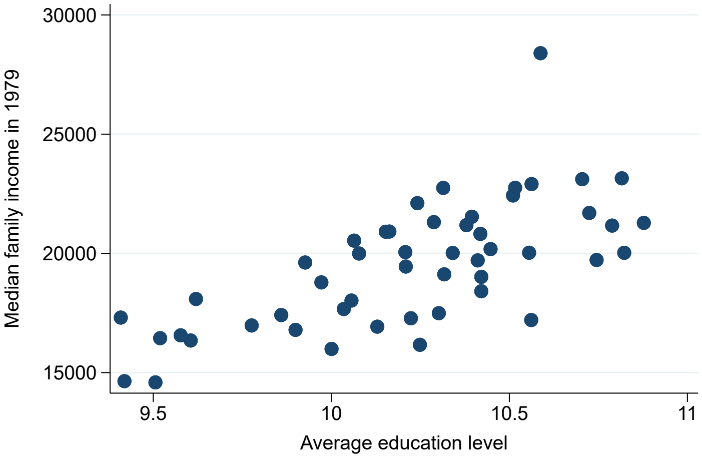
```
twoway scatter faminc educ
```

This graph shows the relationship between one x variable, `educ`, and one y variable, `faminc`. *Uses allstatesdc.dta*



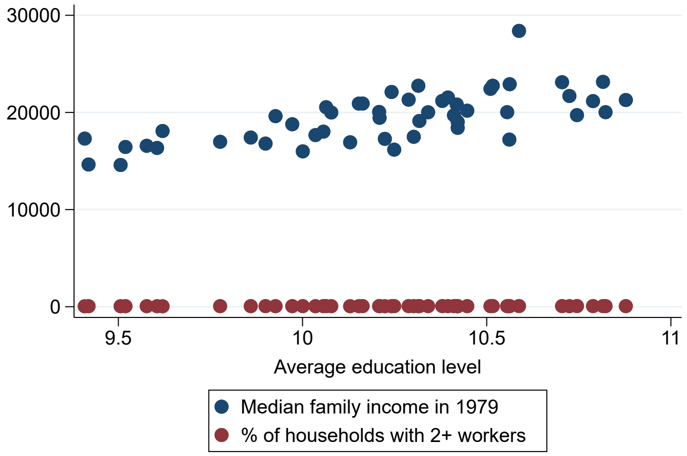
```
twoway (scatter faminc educ, xaxis(1) yaxis(1))
```

By default, the x variable is placed on the first x axis, and the y variable is placed on the first y axis. It is as though we had added the options `xaxis(1)` and `yaxis(1)`, as illustrated here. Note the parentheses emphasize that the options `xaxis(1)` and `yaxis(1)` belong to the `scatter` command and are not



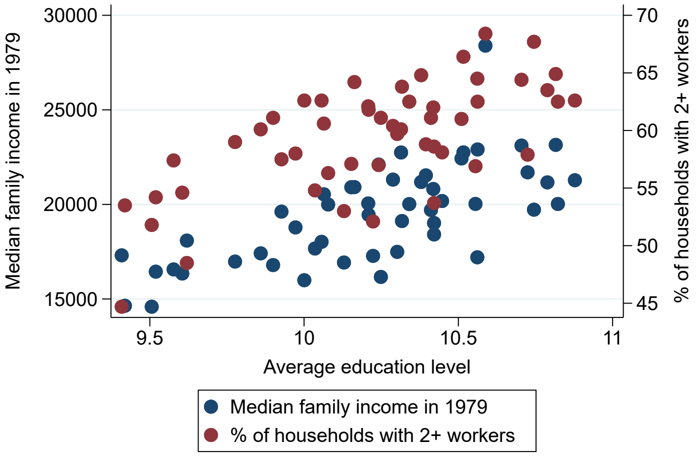
general options for the overall graph, which would appear after the parentheses. *Uses allstatesdc.dta*

```
twoway (scatter faminc educ) (scatter workers2 educ),
legend(cols(1))
```



Now let's overlay a second scatterplot showing `workers2` by `educ`, which has the effect of adding a second variable to the y axis. Stata assumes that all variables are on the first (and thus the same) axis, unless we specify otherwise. As a result, this graph is hard to read because `faminc` is scaled differently from `workers2` but scaled on the same axis. *Uses allstatesdc.dta*

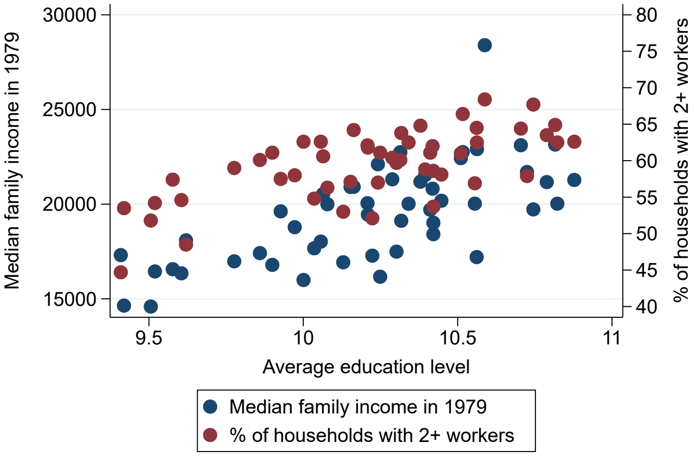
```
twoway (scatter faminc educ, yaxis(1)) (scatter workers2
educ, yaxis(2)), legend(cols(1))
```



Stata permits us to have multiple axes for the x variables and the y variables. In this example, we use the `yaxis(1)` option to place `faminc` on the first y axis and the `yaxis(2)` option to place `workers2` on the second y axis. To make the graph more readable, Stata moved the second y axis to the right side. The `yaxis(1)` option was not needed but was included for clarity. *Uses allstatesdc.dta*

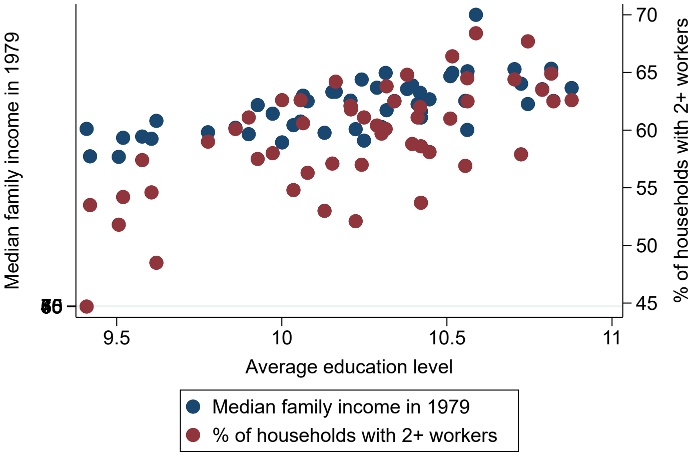
```
twoway (scatter faminc educ) (scatter workers2 educ,
yaxis(2)), ylabel(40(5)80, axis(2)) legend(cols(1))
```

To label `workers2` from 40 to 80, incrementing by 5, we would specify `ylabel(40(5)80, axis(2))`, because `workers2` is on the second axis. Without the `axis(2)` option, Stata would assume that we are referring to the first *y* axis and would change the scaling of `faminc`. *Uses allstatesdc.dta*



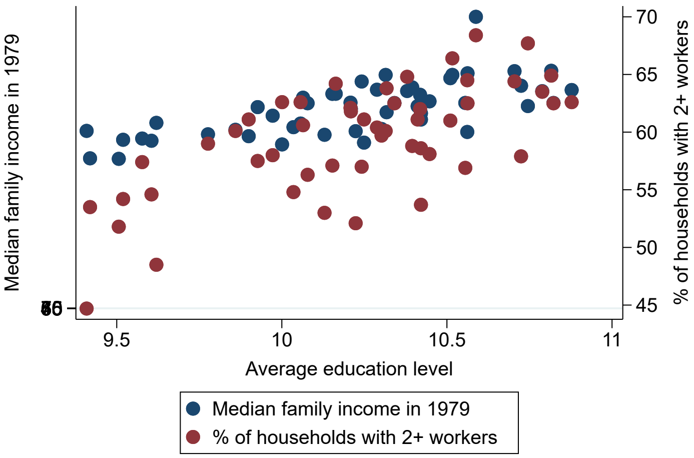
```
twoway (scatter faminc educ) (scatter workers2 educ, yaxis(2)
ylabel(40(5)80)), legend(cols(1))
```

We might be tempted to enter `ylabel()` as an option of the second `scatter` statement and expect the scaling of `workers2` to be modified. However, we can see here that this does not work. This is explained further in the next example. *Uses allstatesdc.dta*



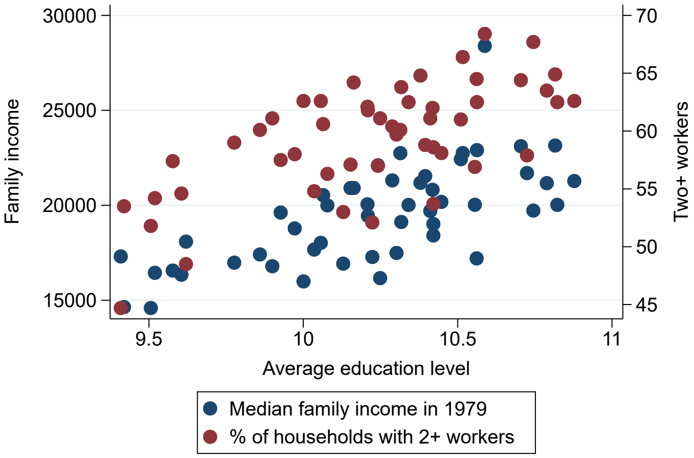
```
twoway (scatter faminc educ) (scatter workers2 educ,
yaxis(2)), ylabel(40(5)80) legend(cols(1))
```

The `ylabel()` option is really a global option. Stata will allow you to specify this option within a plot specification, as shown in the previous example, but the effect will be as though you had typed `ylabel()` as a global option, as shown in this example. Thus, specifying the `ylabel()` option globally is a better alternative. *Uses allstatesdc.dta*



```
twoway (scatter faminc educ) (scatter workers2 educ,
yaxis(2)), ytitle("Family income", axis(1)) ytitle("Two+
workers", axis(2)) legend(cols(1))
```

These same rules apply to modifying the axis titles and labels. In this example, we use the `ytitle()` option to change the titles for the first and second y axes. *Uses allstatesdc.dta*

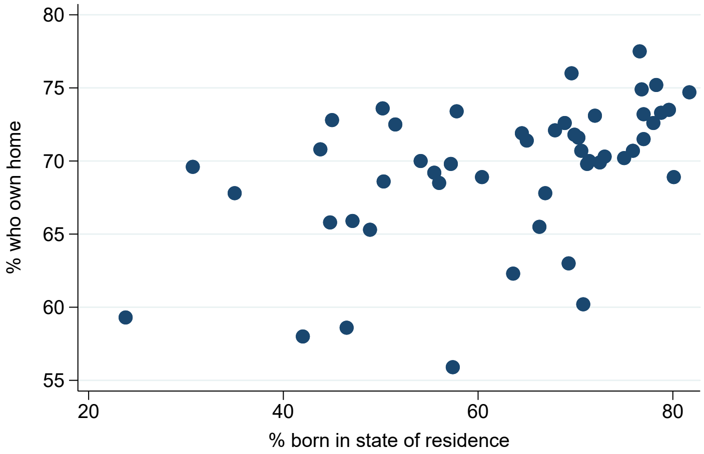


8.8 Graphing by groups

This section provides more details about repeating graphs by using the `by()` option to show separate graphs for each by-group. For more information, see `help by_option`.

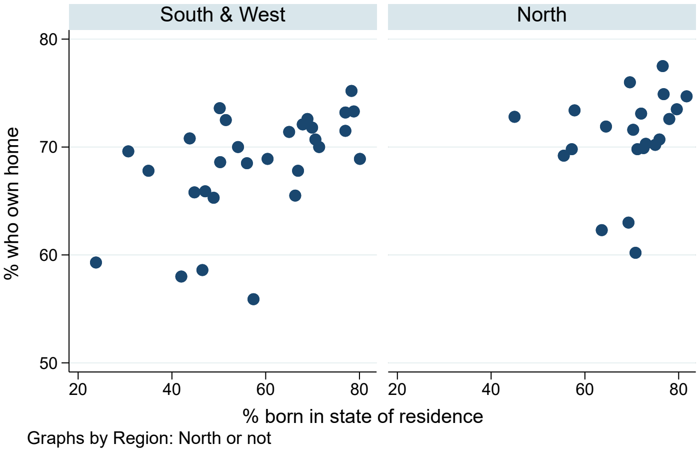
```
twoway scatter ownhome borninstate
```

We start by looking at a scatterplot of `ownhome` and `borninstate`, and we see a general positive relationship such that the higher the percentage of those who were born in the state, the higher the percentage of homeowners in the state. *Uses allstatesdc.dta*



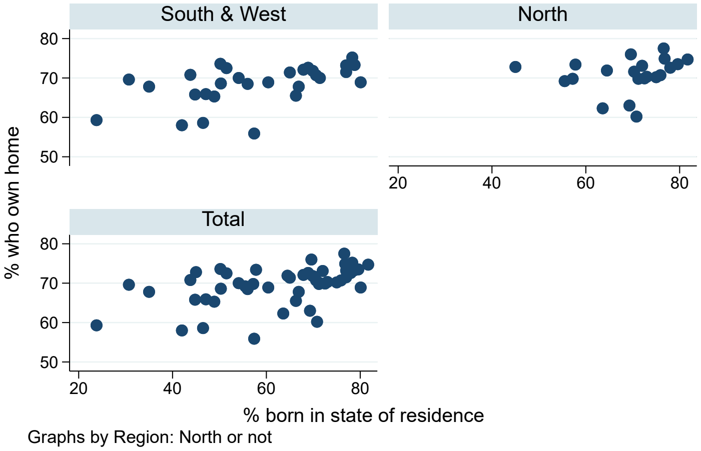
```
twoway scatter ownhome borninstate, by(north)
```

We use the `by(north)` option to look at this relationship broken down by whether the state is considered to be in the North. *Uses allstatesdc.dta*



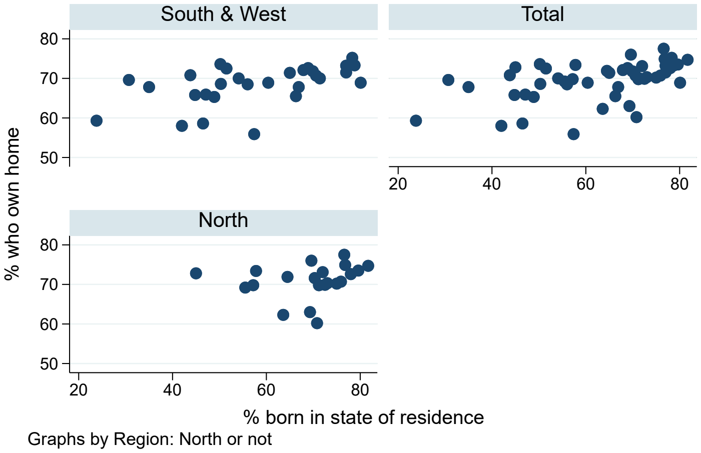
```
twoway scatter ownhome borninstate, by(north, total)
```

We use the `total` option to see the overall relationship for all 50 states, as well as the two plots separately, by the levels of `north`. *Uses allstatesdc.dta*



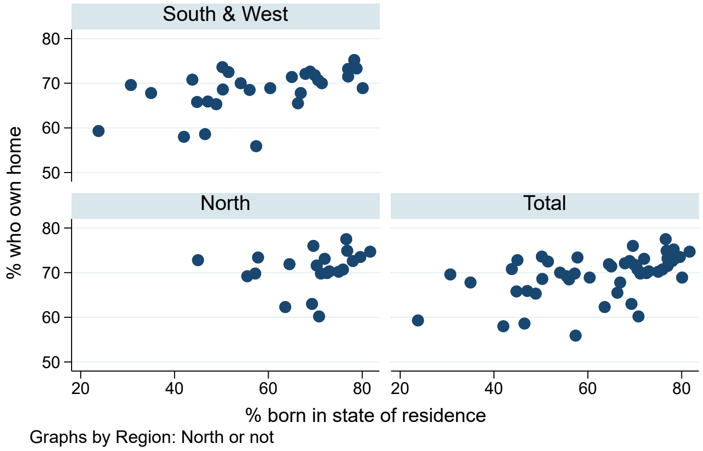
```
twoway scatter ownhome borninstate, by(north, total colfirst)
```

We add the `colfirst` option to show the graphs going down columns first rather than going across rows first, which is the default. *Uses allstatesdc.dta*



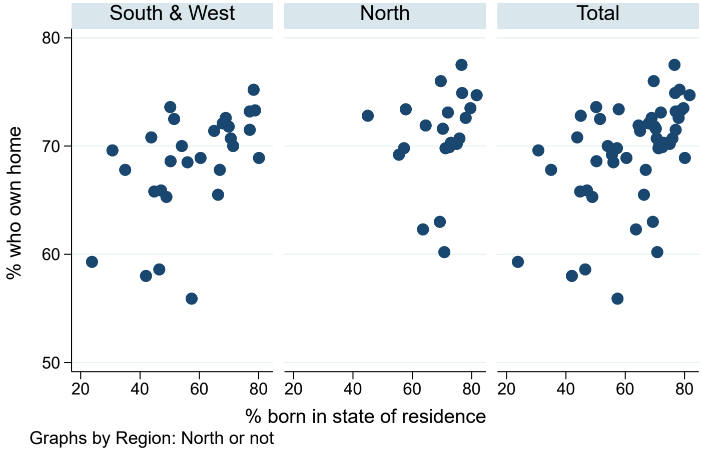
```
twoway scatter ownhome borninstate, by(north, total holes(2))
```

The `holes(2)` option leaves the second position empty. Here we specify one position to leave empty, but we can specify multiple positions within the `holes()` option. *Uses allstatesdc.dta*



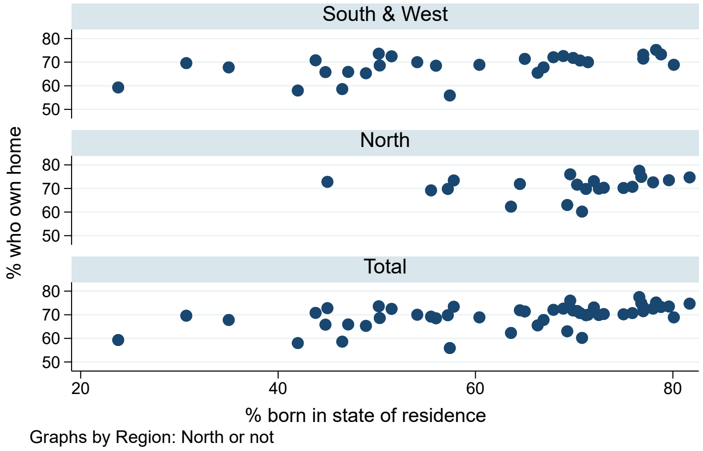
```
twoway scatter ownhome borninstate, by(north, total rows(1))
```

The `rows(1)` option indicates that the graph should be displayed in one row. *Uses allstatesdc.dta*



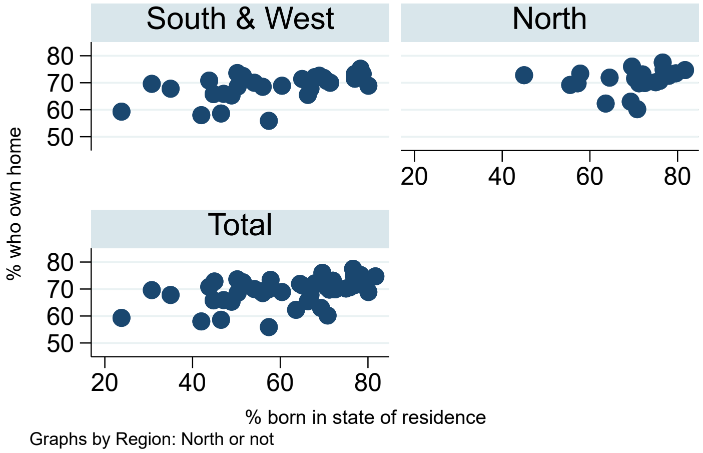
```
twoway scatter ownhome borninstate, by(north, total cols(1))
```

The `cols(1)` option shows the graph in one column. *Uses allstatesdc.dta*



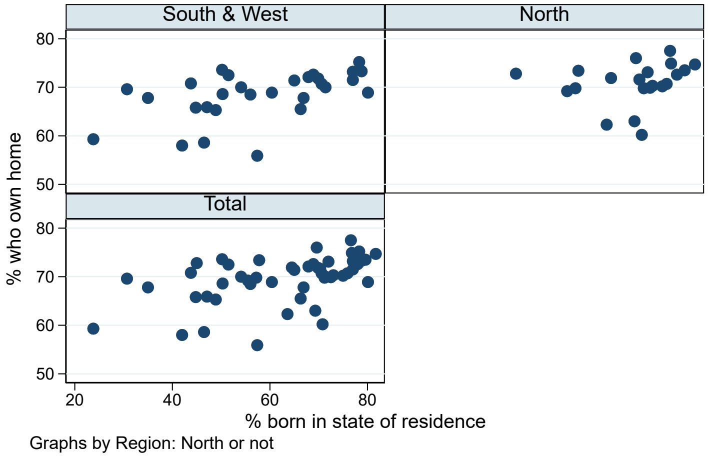
```
twoway scatter ownhome borninstate, by(north, total
iscale(*1.5))
```

Sometimes with the `by()` option, the graphs become small, making the text and symbols difficult to see. The `iscale()` option magnifies the size of these elements. Here we increase the size of these elements by a factor of 1.5. *Uses allstatesdc.dta*

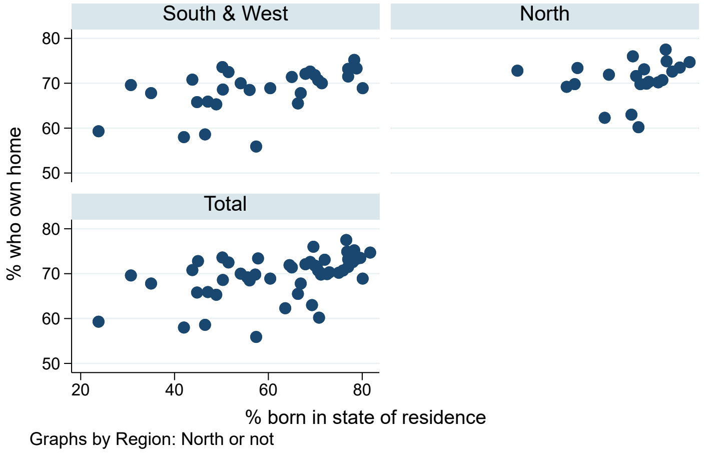


```
twoway scatter ownhome borninstate, by(north, total compact)
```

The `compact` option displays the graph using a compact style, pushing the graphs tightly together. This is almost the same as specifying `style(compact)`. *Uses allstatesdc.dta*



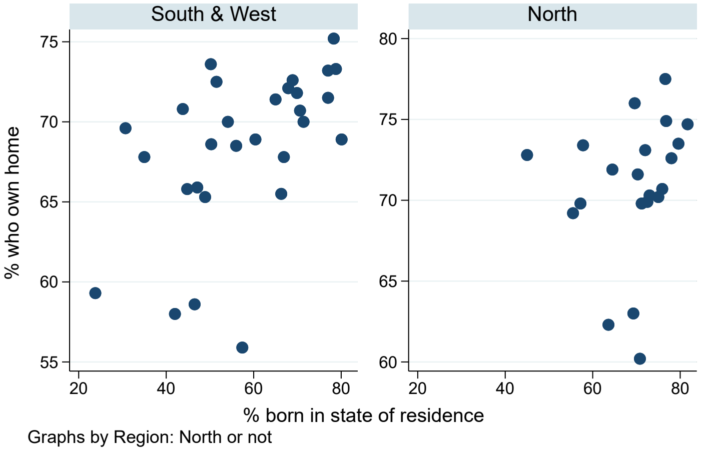
```
twoway scatter ownhome borninstate, by(north, total
noedgelabel)
```



The `noedgelabel` option suppresses the display of the x axis for the graphs that do not appear on the bottom row, in this case, the graph for the North. *Uses allstatesdc.dta*

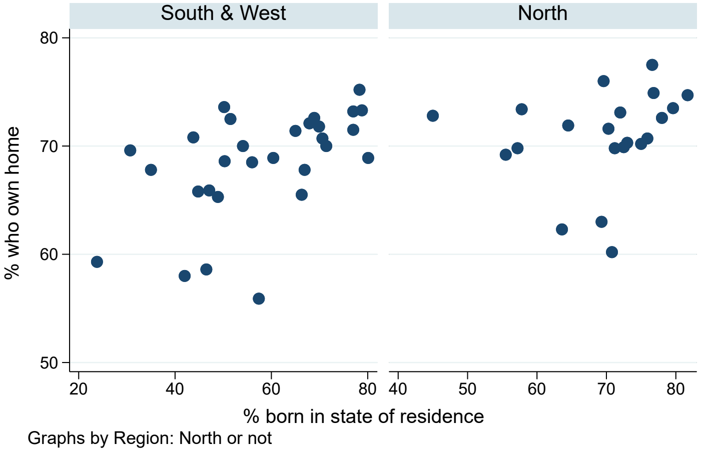
```
twoway scatter ownhome borninstate, by(north, yrescale)
```

The `yrescale` option allows the y variables to be scaled independently for each by-group. *Uses allstatesdc.dta*



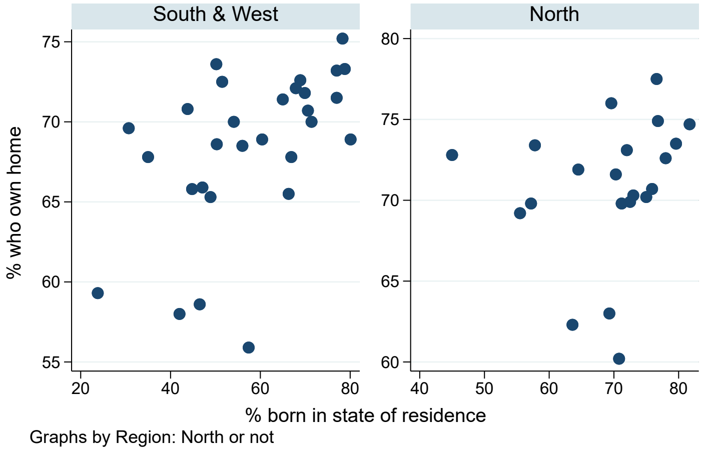
```
twoway scatter ownhome borninstate, by(north, xrescale)
```

Likewise, the `xrescale` option allows the x variable to be scaled differently across all the by-groups. *Uses allstatesdc.dta*



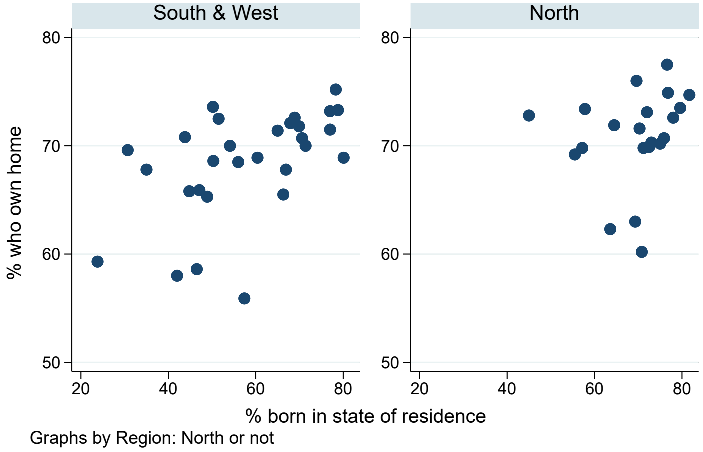
```
twoway scatter ownhome borninstate, by(north, rescale)
```

The `rescale` option scales both the x variable and y variable differently across the by-groups. Both axes are separately rescaled. *Uses allstatesdc.dta*



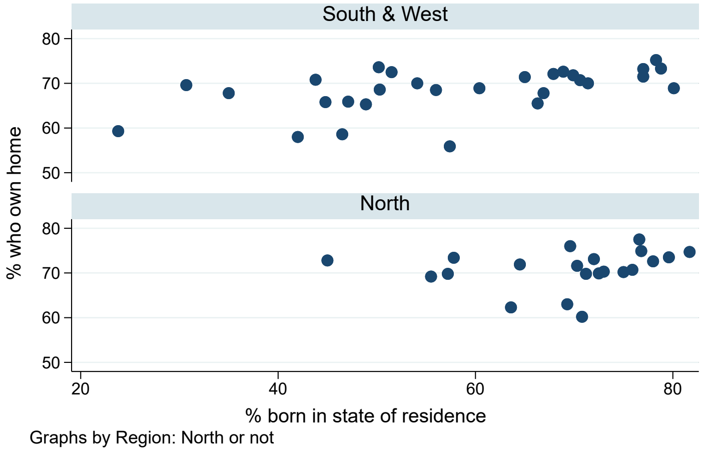
```
twoway scatter ownhome borninstate, by(north, iyaxes)
```

The `iyaxes` option displays the y axes for each individual graph. *Uses allstatesdc.dta*



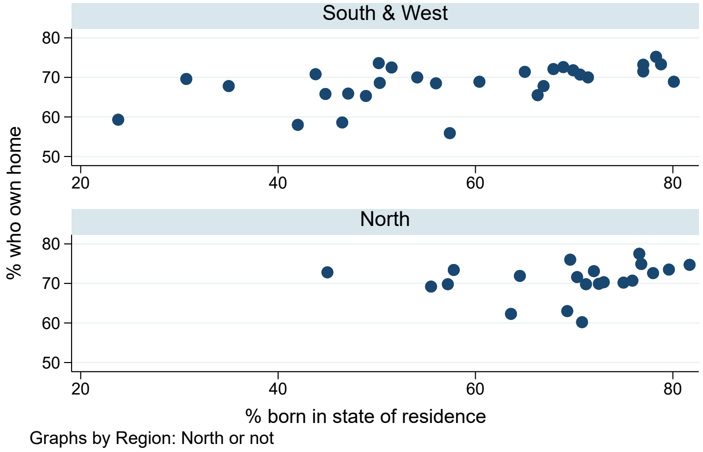
```
twoway scatter ownhome borninstate, by(north, cols(1))
```

Likewise, the `ixaxes` option displays the x axis for all graphs. Here we omit this option and instead use the `cols(1)` option to display two graphs in one column. Stata omits the x axis in the top graph. *Uses allstatesdc.dta*



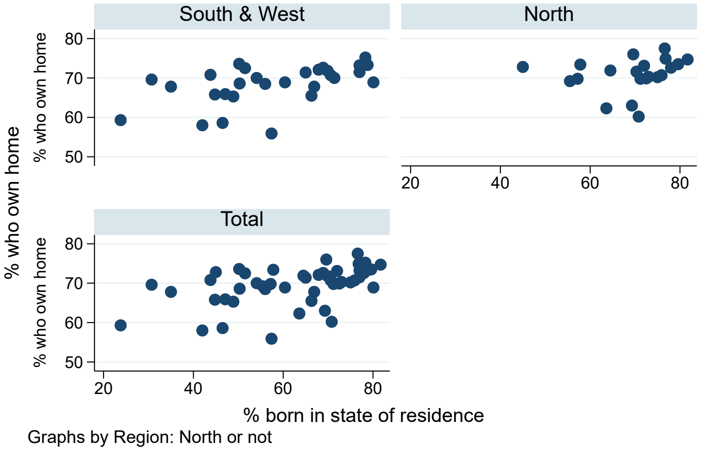
```
twoway scatter ownhome borninstate, by(north, ixaxes cols(1))
```

We now include the `ixaxes` option and see that the x axis is now displayed on the top graph. *Uses allstatesdc.dta*



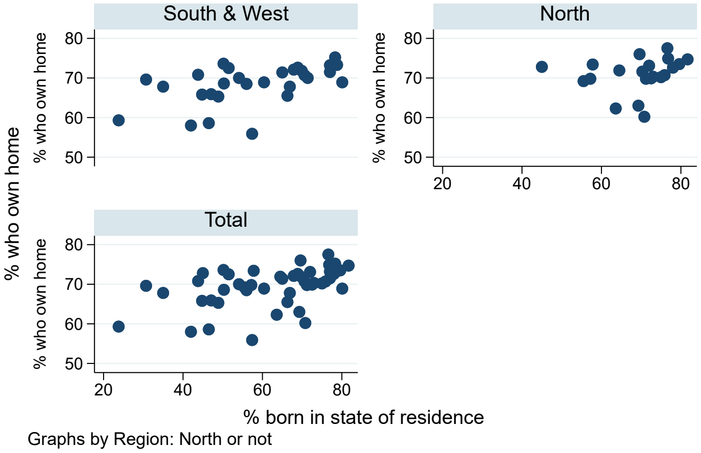
```
twoway scatter ownhome borninstate, by(north, total iytitle)
```

We display the title for each y axis by using the `iytitle` option. *Uses allstatesdc.dta*



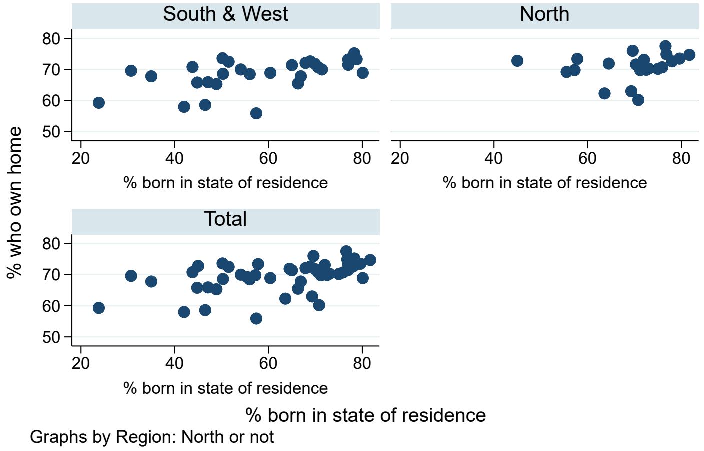
```
twoway scatter ownhome borninstate, by(north, total iyaxes
iytitle)
```

The y title was not displayed for the North because the y axis was omitted for that graph. We now include the `iyaxes` and `iytitle` options to display the y axis and y title for the North graph as well. *Uses allstatesdc.dta*

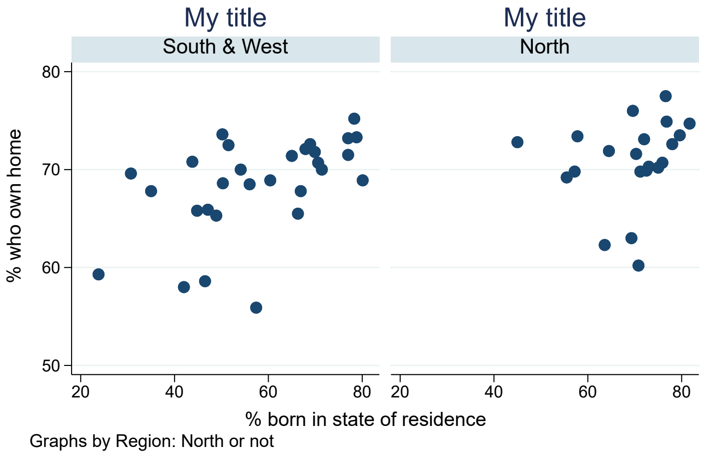


```
twoway scatter ownhome borninstate, by(north, total ixaxes
ixtitle)
```

Likewise, we can display the x title on each graph using the `ixaxes` and `ixtitle` options. *Uses allstatesdc.dta*



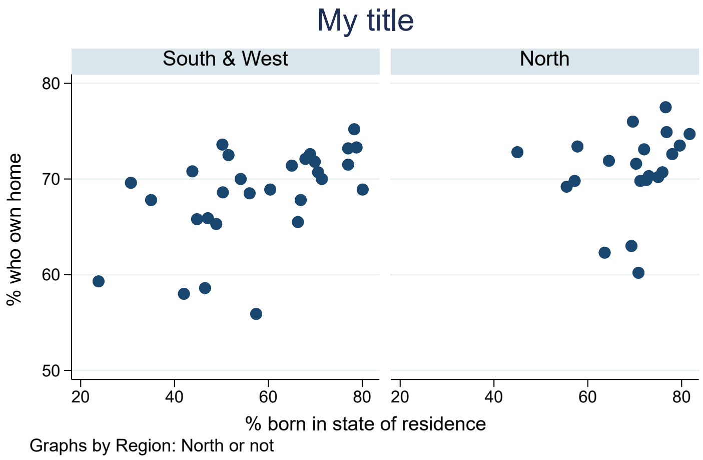
```
twoway scatter ownhome borninstate, by(north) title("My
title")
```



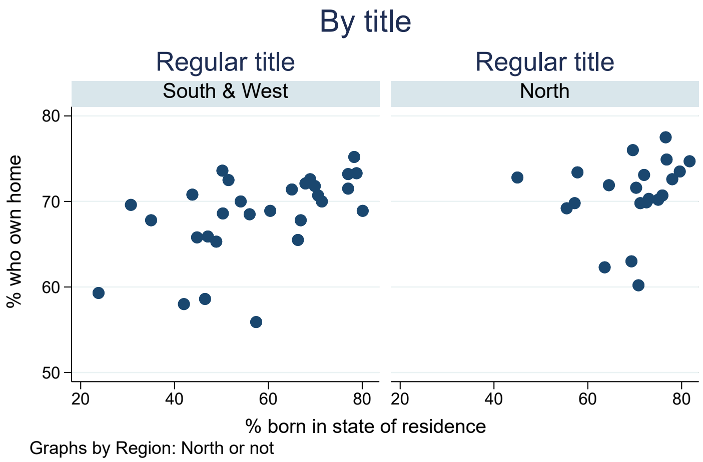
If we include a `title()` option with `by()`, Stata creates each graph separately using the title we specify. *Uses allstatesdc.dta*

```
twoway scatter ownhome borninstate, by(north, title("My
title"))
```

If we add `title()` within the `by()` option, Stata makes this an overall title for the graph. *Uses allstatesdc.dta*



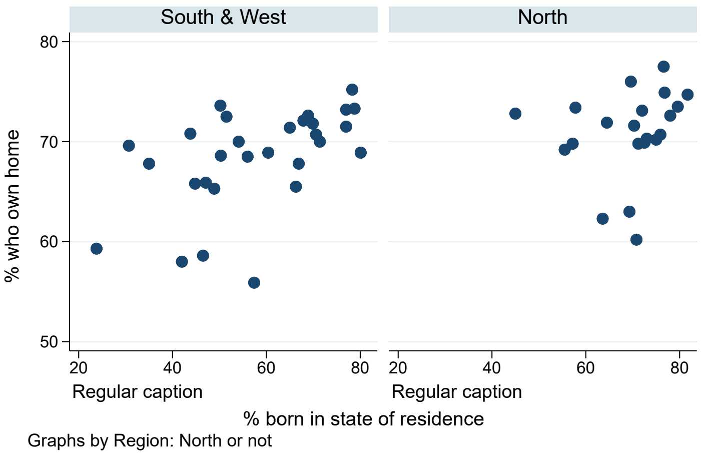
```
twoway scatter ownhome borninstate, by(north, title("By
title")) title("Regular title")
```



This example should help you to understand how these two types of titles work. When we use the `title()` option outside the `by()` option, it applies to all graphs that are created because it is repeated with the `by()` option. The `by(title())` option is applied after all smaller graphs are created, providing a title for the entire set of subgraphs. *Uses allstatesdc.dta*

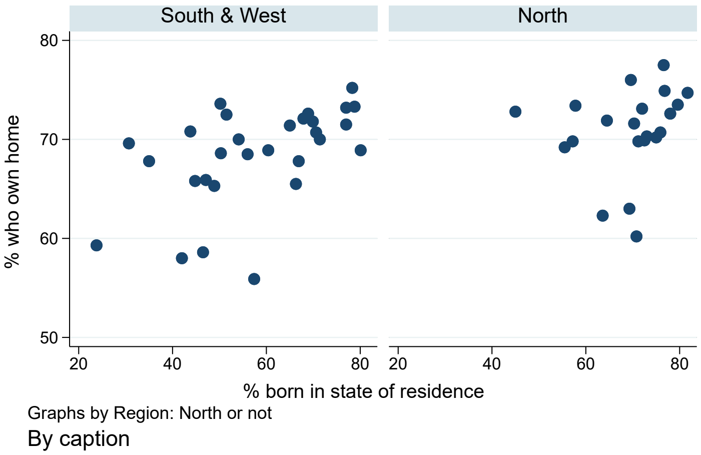
```
twoway scatter ownhome borninstate, by(north)
caption("Regular caption")
```

Stata treats the `caption()` option the same way that it treats titles. Here we include the `caption()` option outside the `by()` option. The caption is thus displayed with each graph. *Uses allstatesdc.dta*



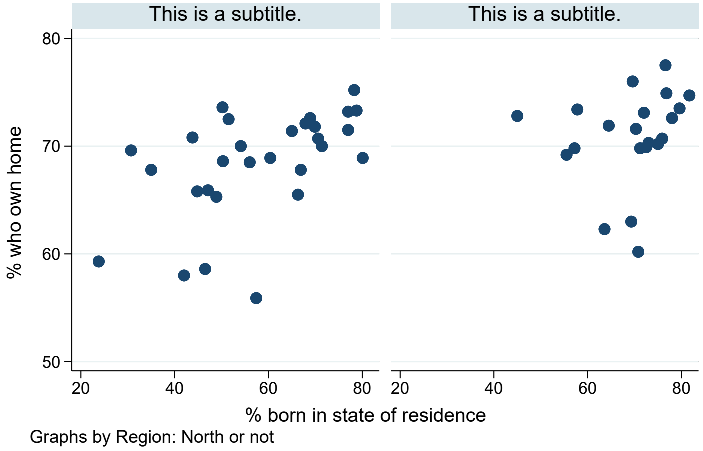
```
twoway scatter ownhome borninstate, by(north, caption("By caption"))
```

When we include the `caption()` option inside the `by()` option, it is displayed as a caption for the full graph. *Uses allstatesdc.dta*



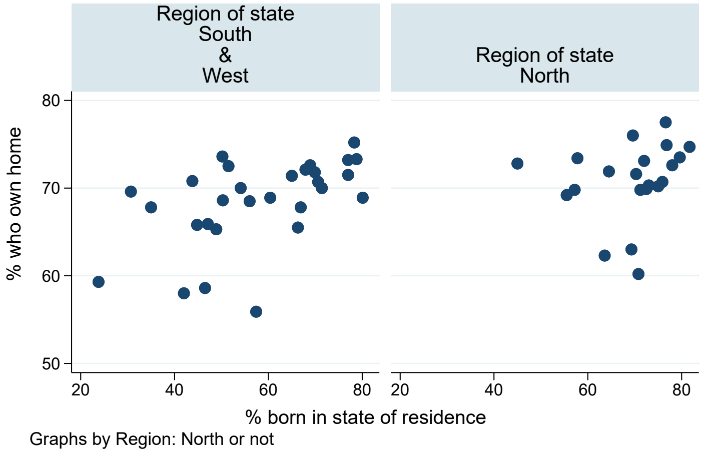
```
twoway scatter ownhome borninstate, by(north) subtitle("This is a subtitle.")
```

Stata treats the `subtitle()` option differently than the `title()` and `caption()` options. Here we include a `subtitle()` option, and we see that the subtitle has replaced the title above each graph that represented the names of the by-group. *Uses allstatesdc.dta*



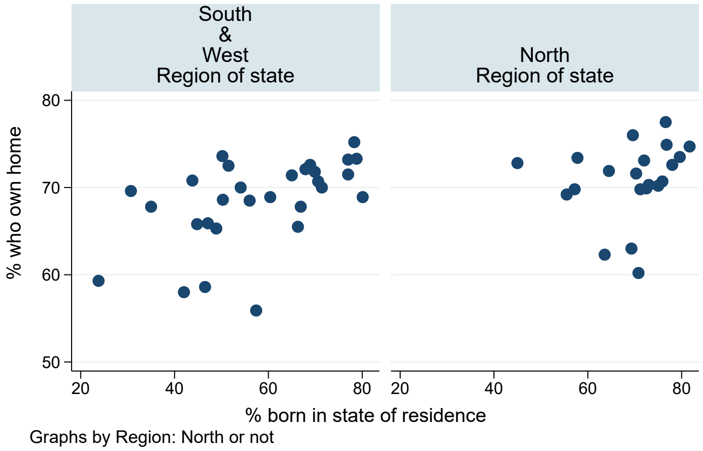
```
twoway scatter ownhome borninstate, by(north)
subtitle("Region of state", prefix)
```

We can use the `subtitle()` option to add more labeling to the by-group names. Here we use the `prefix` option to insert text that appears in the subtitle before the name of the by-group. *Uses allstatesdc.dta*



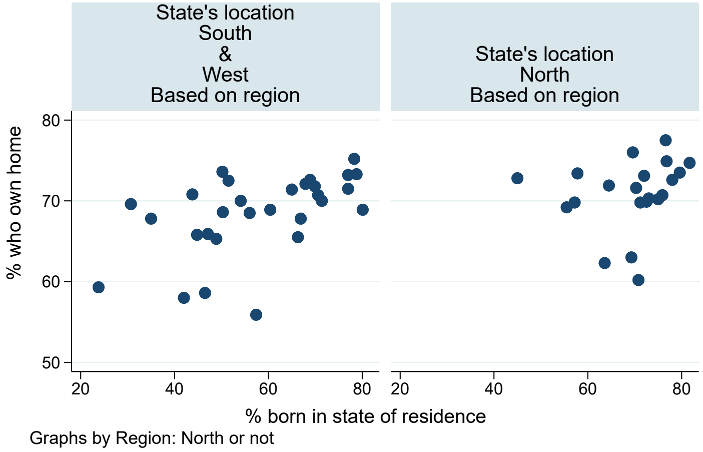
```
twoway scatter ownhome borninstate, by(north)
subtitle("Region of state", suffix)
```

We can use the `suffix` option to insert text that appears in the subtitle after the name of the by-group. *Uses allstatesdc.dta*



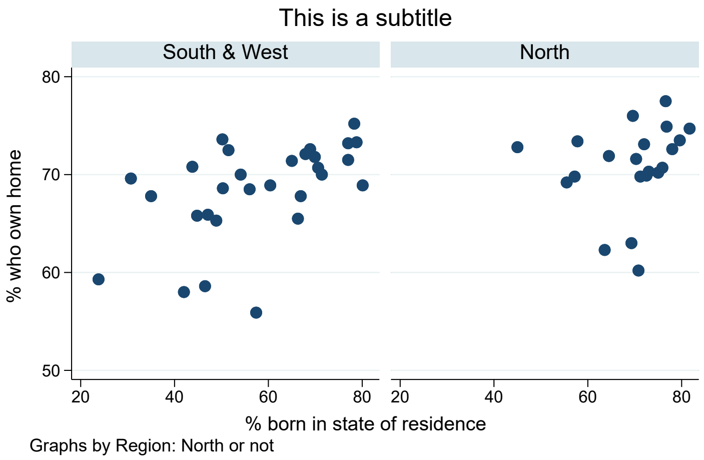
```
twoway scatter ownhome borninstate, by(north)
subtitle("State’s location", prefix) subtitle("Based on
region", suffix)
```

We can even combine the `prefix` and `suffix` options to insert text before and after the label of the by-group. *Uses allstatesdc.dta*



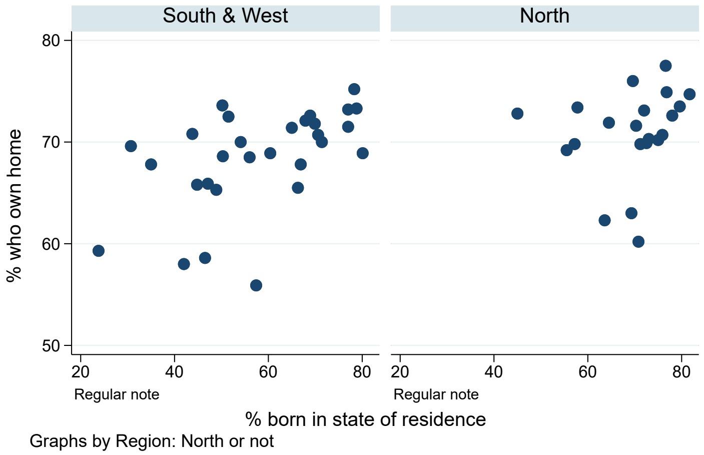
```
twoway scatter ownhome borninstate, by(north, subtitle("This
is a subtitle"))
```

When used within the `by()` option, the `subtitle()` option works just like the `title()` and `caption()` options, placing a subtitle on the entire graph. *Uses allstatesdc.dta*



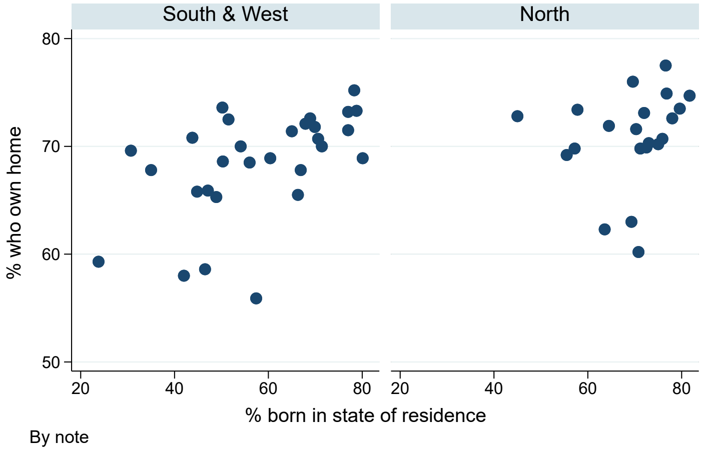
```
twoway scatter ownhome borninstate, by(north) note("Regular
note")
```

Stata treats the `note()` option much as it does the `title()`, `caption()`, and `subtitle()` options. Here we include a `note()` option and see that it is shown below both graphs. *Uses allstatesdc.dta*



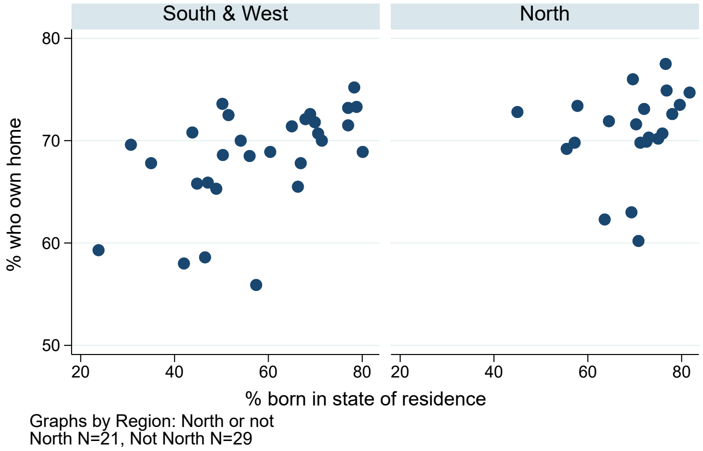
```
twoway scatter ownhome borninstate, by(north, note("By
note"))
```

If we include `note()` within the `by()` option, we see that our note overrides the note that Stata provided to indicate that the graphs were separated by the variable `north`. *Uses allstatesdc.dta*



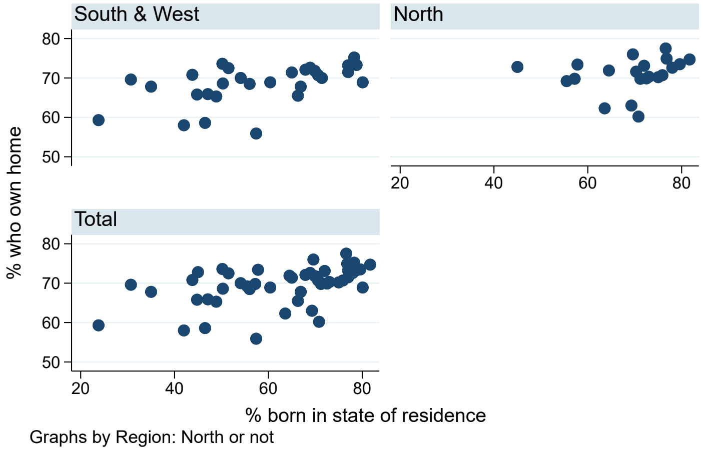
```
twoway scatter ownhome borninstate, by(north, note("North
N=21, Not North N=29", suffix))
```

As shown with the `subtitle()` option, the `prefix` or `suffix` option adds text before or after the existing note. *Uses allstatesdc.dta*

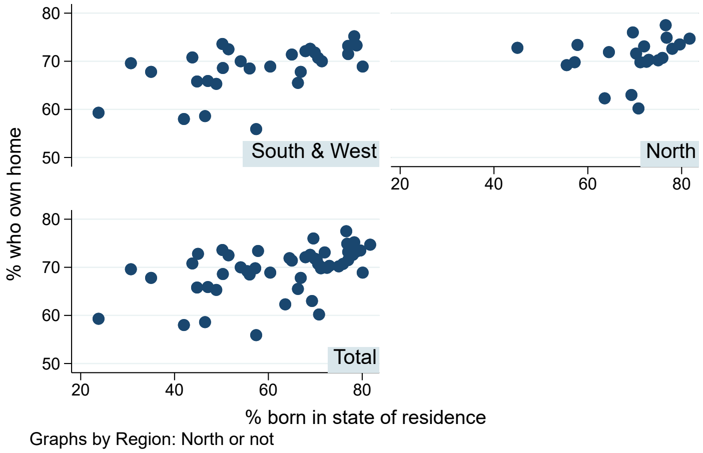


```
twoway scatter ownhome borninstate, by(north, total)
subtitle(, position(11))
```

Previously, we saw that we could use the `subtitle()` option to modify the by-group names above each graph. Here we use the `subtitle(, position(11))` option to modify the placement of this text to appear in the eleven o’clock position. *Uses allstatesdc.dta*



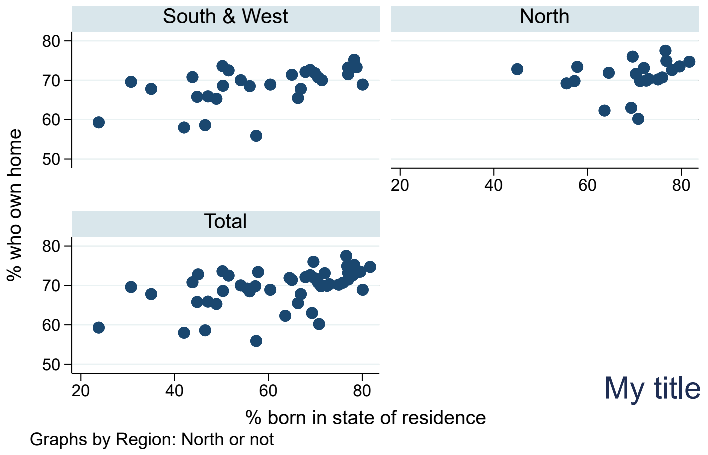
```
twoway scatter ownhome borninstate, by(north, total)
subtitle(, position(5) ring(0) nobexpand)
```



We can place the name of the by-group in the bottom right corner of each graph by using the `subtitle()` option. The options `position(5)` and `ring(0)` move the subtitle to the five o'clock position and inside the plot region. The `nobexpand` (no box expand) option prevents the by-group name from expanding to consume the entire plot region. *Uses allstatesdc.dta*

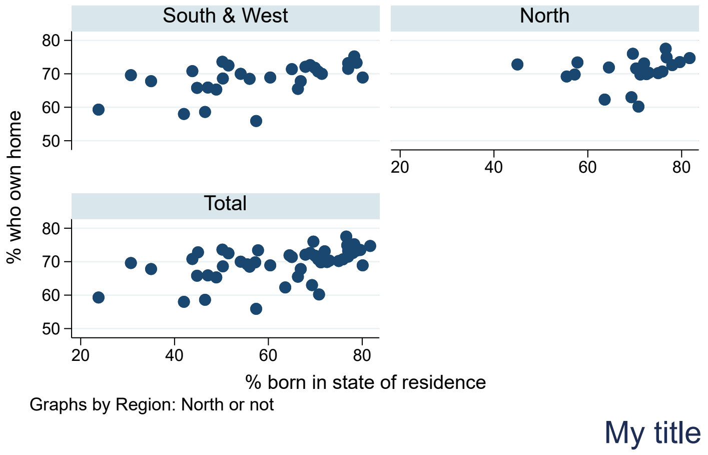
```
twoway scatter ownhome borninstate, by(north, total title("My
title", ring(0) position(5)))
```

We can also use the `ring()` and `position()` options with `title()`, `note()`, and `caption()`. Here we use `position(5)` to put the title in the bottom right corner and `ring(0)` to locate it inside the plot region. *Uses allstatesdc.dta*



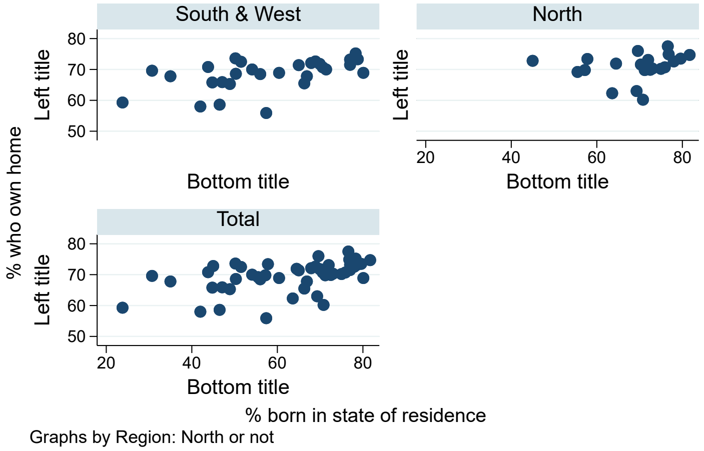
```
twoway scatter ownhome borninstate, by(north, total title("My
title", position(5)))
```

We repeat the previous command without the `ring(0)` option to illustrate that without this option the title is placed outside the plot region. *Uses allstatesdc.dta*



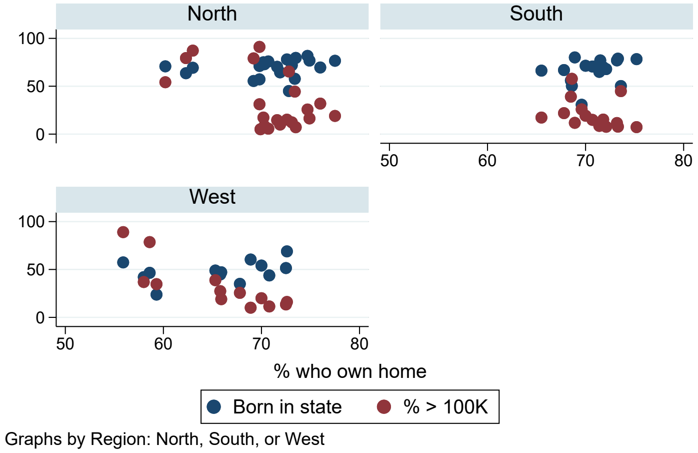
```
twoway scatter ownhome borninstate, by(north, total)
l1title("Left title") b1title("Bottom title")
```

Including the `l1title()` option adds a title to the left (on the y axis) of each of the graphs. Likewise, the `b1title()` option adds a title to the bottom (on the x axis) of each of the graphs. *Uses allstatesdc.dta*



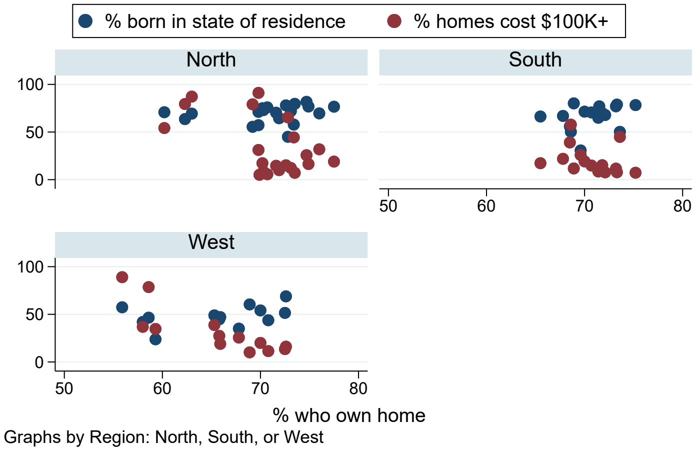
```
twoway scatter (borninstate propval100 ownhome), by(nsw)
legend(label(1 "Born in state") label(2 "% > 100K"))
```

Here we use the `legend()` option to change the labels associated with the first two keys. These options modify the contents of the legend, so they should appear outside the `by()` option. *Uses allstatesdc.dta*



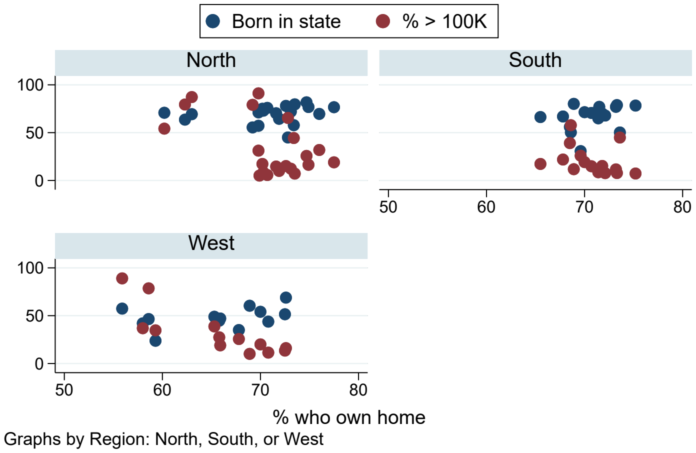
```
twoway scatter (borninstate propval100 ownhome), by(nsw,
legend(position(12)))
```

In this graph, we use the `position()` option to modify the position of the legend. Note that options that modify the position of the legend need to be placed within the `by()` option. *Uses allstatesdc.dta*



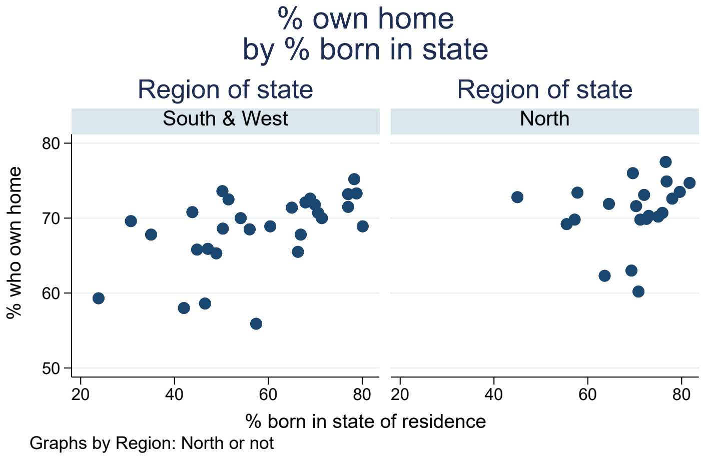
```
twoway scatter (borninstate propval100 ownhome), by(nsw,
legend(position(12))) legend(label(1 "Born in state") label(2
"% > 100K"))
```

Here we use both options from the previous two graphs—the `legend()` option is used twice: inside the `by()` option to modify the legend's position and outside the `by()` option to modify the legend's contents. The use of `legend()` with the `by()` option is covered more in Options : Legend (section 8.9). *Uses allstatesdc.dta*

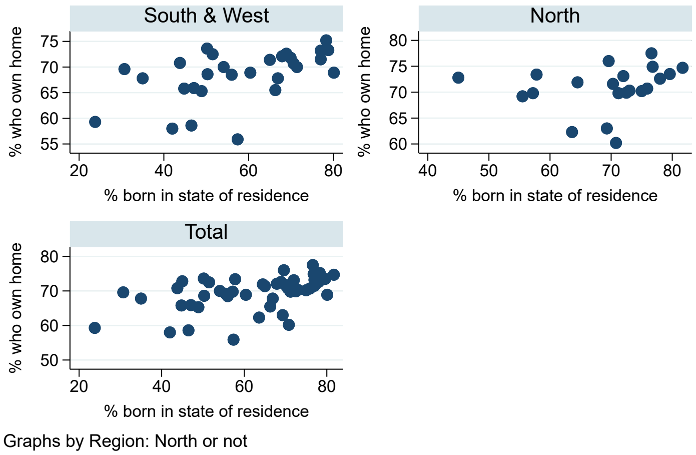


```
twoway scatter ownhome borninstate, by(north, title("% own
home" "by % born in state")) title("Region of state")
```

The `title()` option within the `by()` option makes a title for the entire graph, whereas the second `title()` option makes a title that is displayed for each graph. *Uses allstatesdc.dta*



```
twoway scatter ownhome borninstate, by(north, total rescale
ixtitle iytitle b1title("") l1title(""))
```



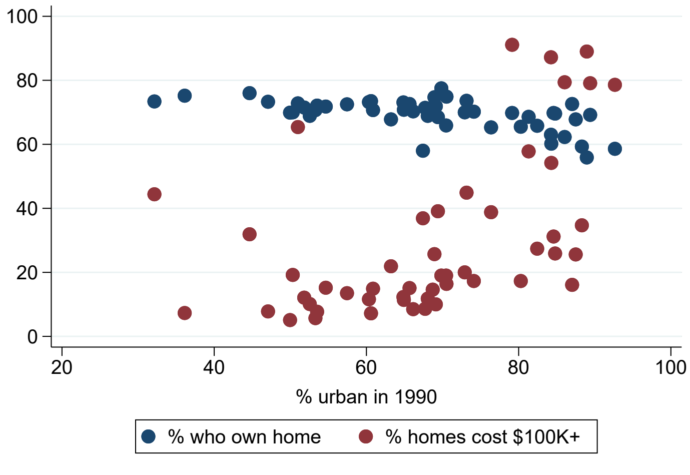
Here we obtain separate graphs for the three groups, using `rescale` to obtain different x- and y-axis labels and scales, `ixtitle` and `iytitle` to title the graphs separately, and `b1title()` and `l1title()` to suppress the overall titles for the x and y axes. *Uses allstatesdc.dta*

8.9 Controlling the legend

This section describes more details about using the legend. Legends can be useful in several situations, and this section shows how to customize them. For more information about legend options, see `help legend_options`. Also, for controlling the text and textbox of the legend, see Options : Textboxes (section 8.11) and Options : Adding text (section 8.10).

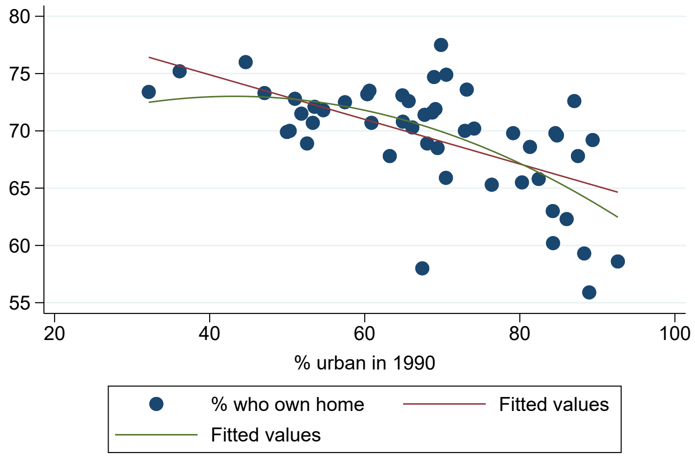
```
twoway scatter ownhome propval100 urban
```

Legends can be created in many ways. For example, here we have two y variables, `ownhome` and `propval100`, on the same plot, and Stata creates a legend labeling the different points. The default legend for this graph is quite useful. *Uses allstatesdc.dta*

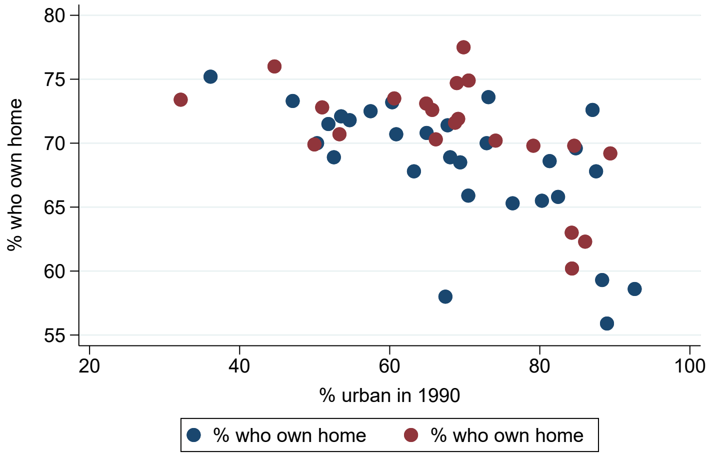


```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban)
```

Legends are also created when we overlay plots. Here Stata adds a legend entry for each overlaid plot. The default legend is less useful because it does not help us differentiate between the kinds of fit values. *Uses allstatesdc.dta*



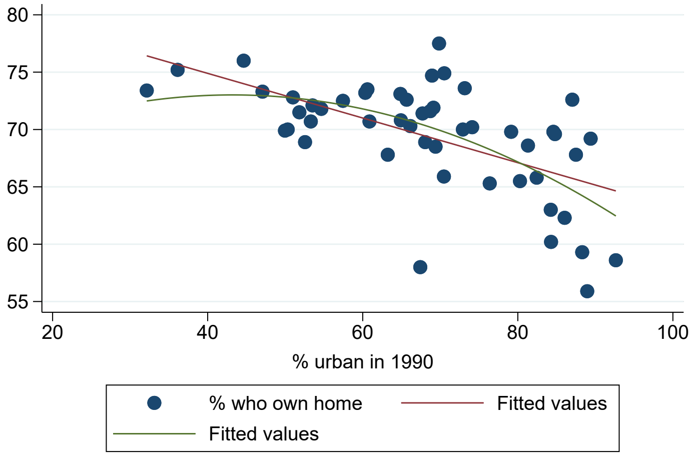
```
twoway (scatter ownhome urban if north==0) (scatter ownhome
urban if north==1)
```



A third example is when we overlay two plots by using `if` to display the same variables but for different observations. Here we show the same scatterplot separately for states in the North and for those not in the North. This legend does not help us differentiate the markers. *Uses allstatesdc.dta*

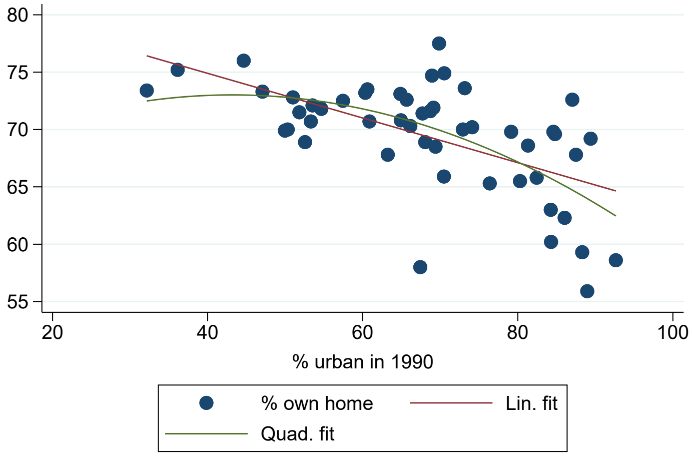
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban)
```

Regardless of the graph command(s) that generated the legend, it can be customized the same way. For many examples, we will use this graph for customizing the legend. *Uses allstatesdc.dta*



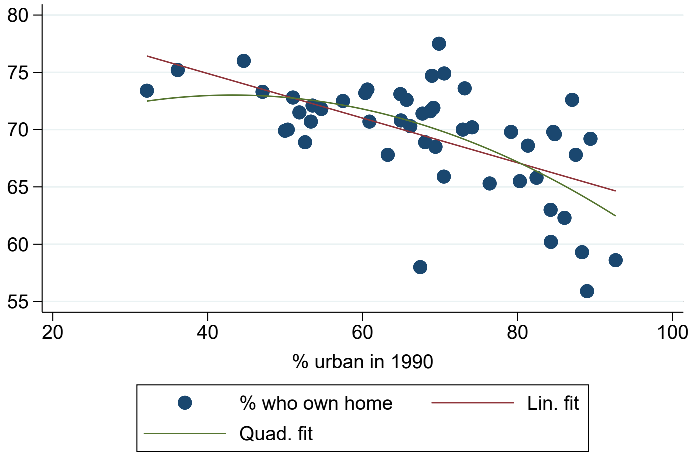
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(label(1 "% own home") label(2 "Lin.
fit") label(3 "Quad. fit"))
```

The `label()` option assigns labels for the keys. Note that we can use a separate `label()` option to modify each key. *Uses allstatesdc.dta*



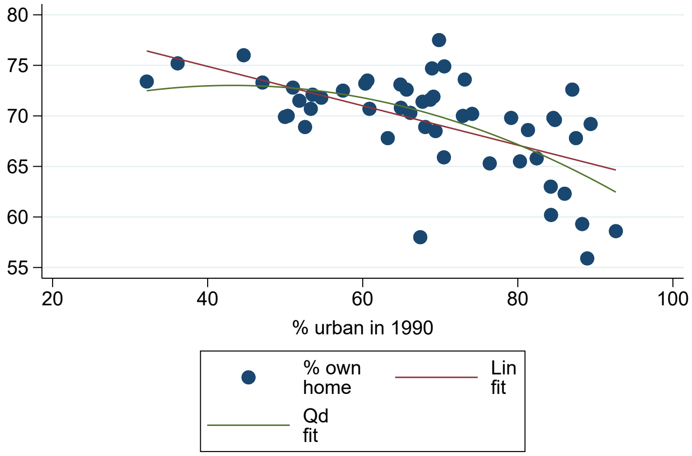
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(label(2 "Lin. fit") label(3 "Quad.
fit"))
```

We can also use the `label()` option to modify some of the keys. Here we modify only the second and third keys. *Uses allstatesdc.dta*



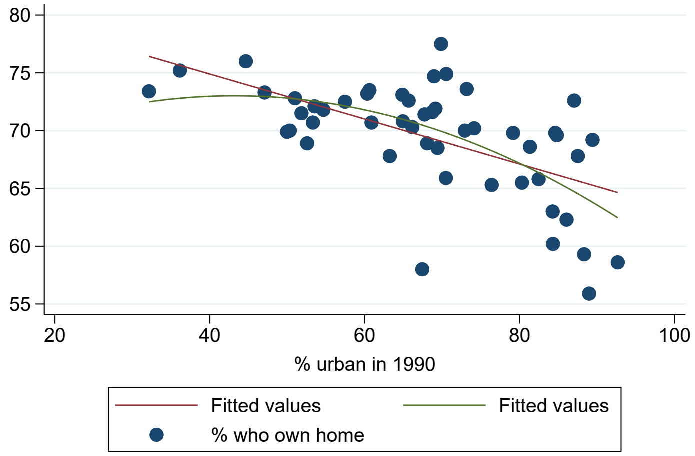
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(label(1 "% own" "home") label(2 "Lin"
"fit") label(3 "Qd" "fit"))
```

We can place the label on multiple lines by including multiple quoted strings. *Uses allstatesdc.dta*



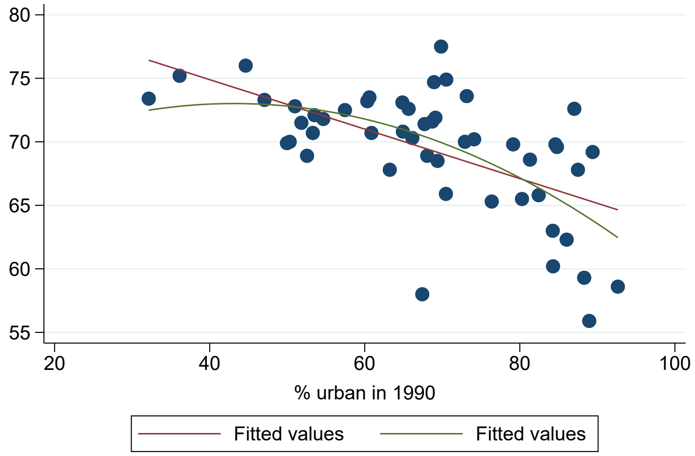
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(2 3 1))
```

The `order()` option changes the order of the keys in the legend. *Uses allstatesdc.dta*



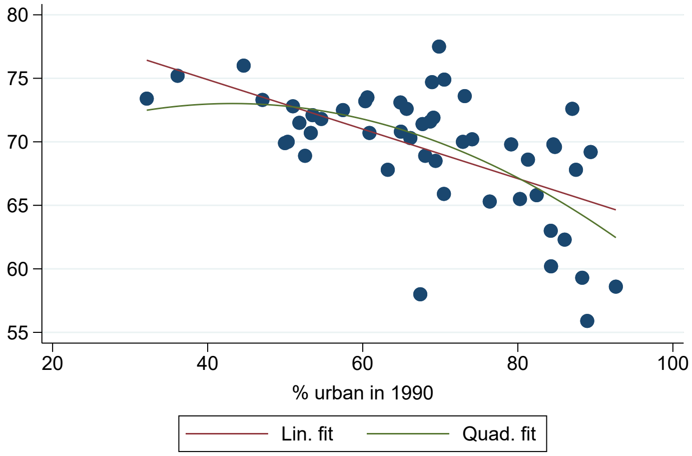
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(2 3))
```

We can also omit keys from the `order()` option to suppress their display in the legend. Here we suppress the display of the first key. *Uses allstatesdc.dta*



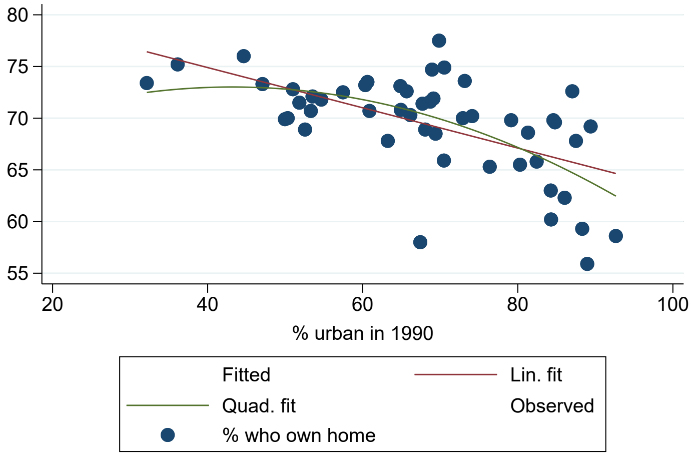
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(2 "Lin. fit" 3 "Quad. fit"))
```

We can also insert and replace text for the keys with the `order()` option. Here we hide the first key and replace the text for keys 2 and 3. *Uses allstatesdc.dta*



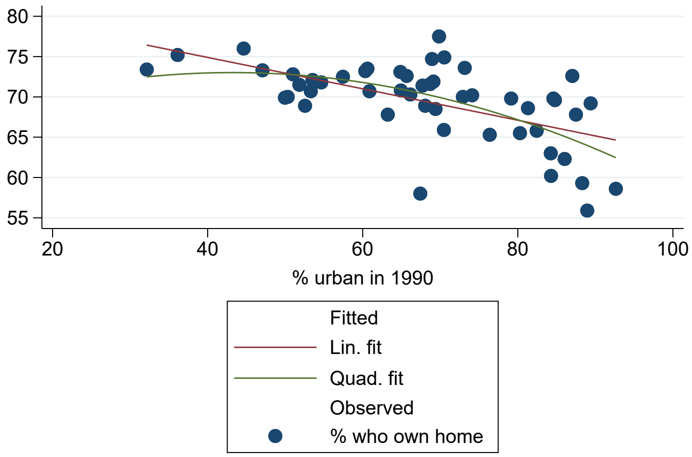
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Fitted" 2 "Lin. fit" 3 "Quad.
fit" - "Observed" 1))
```

We use `- "Fitted"` to insert the word *Fitted* and `- "Observed"` to insert the word *Observed*. Because of the organization of the keys in the legend, the example of this option is hard to follow. *Uses allstatesdc.dta*



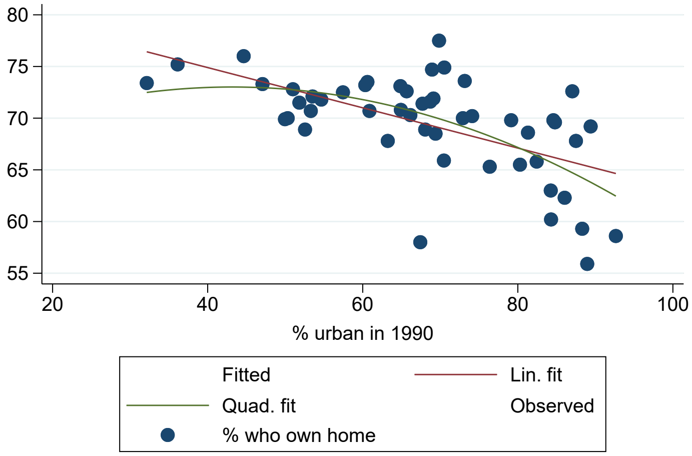
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Fitted" 2 "Lin. fit" 3 "Quad.
fit" - "Observed" 1) cols(1))
```

The `cols()` option displays the legend in one column. Here the added text makes more sense, but the legend uses quite a bit of space. *Uses allstatesdc.dta*



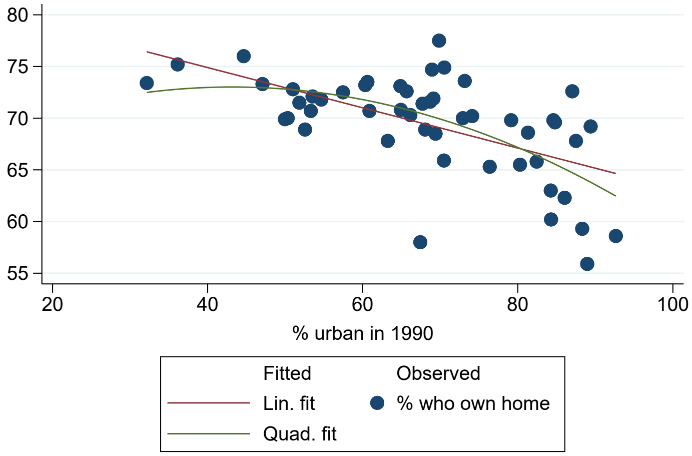
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Fitted" 2 "Lin. fit" 3 "Quad.
fit" - "Observed" 1) rows(3))
```

Here we use the `rows()` option to display the legend in three rows. The next example shows how we can display the fitted keys in the left column and the observed keys in the right column. *Uses allstatesdc.dta*



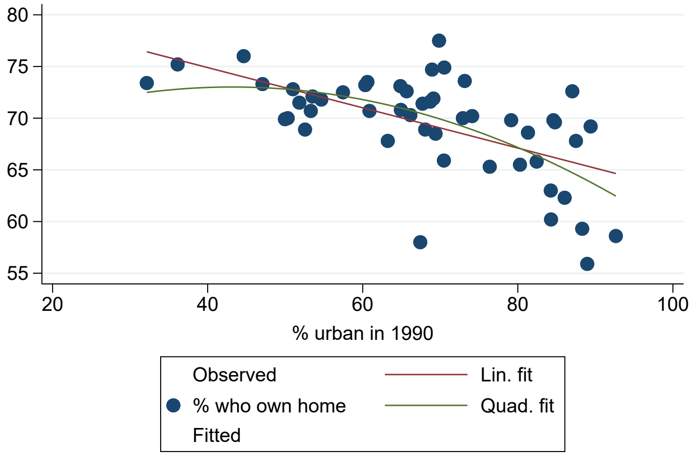
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Fitted" 2 "Lin. fit" 3 "Quad.
fit" - "Observed" 1) rows(3) colfirst)
```

Adding the `colfirst` option displays the keys in column order instead of row order, with the *Fitted* keys in the left column and the *Observed* keys in the right column. *Uses allstatesdc.dta*



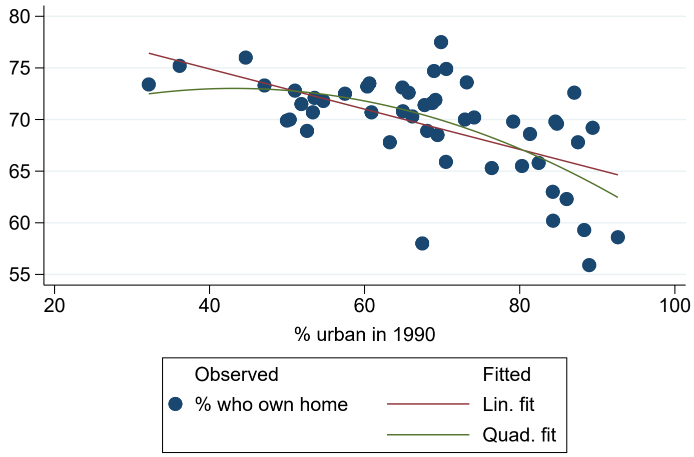
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Observed" 1 - "Fitted" 2
"Lin. fit" 3 "Quad. fit") rows(3) colfirst)
```

This legend is the same as the one in the previous example, but here we attempt to place the *Observed* keys in the left column and the *Fitted* keys in the right column. However, the word *Fitted* appears at the bottom of the first column instead of the top of the second column. *Uses allstatesdc.dta*



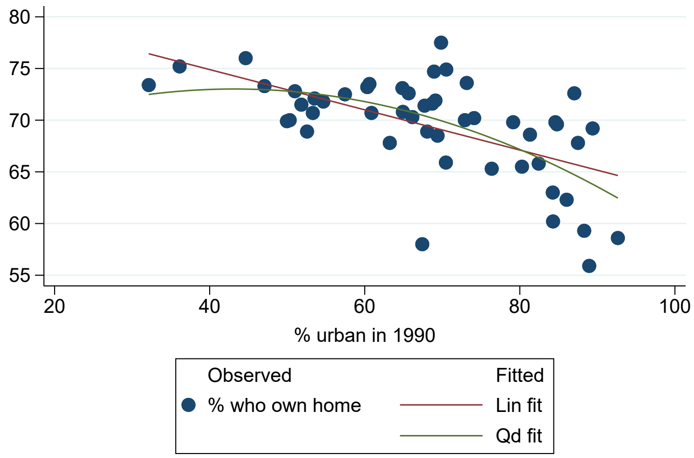
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Observed" 1 - "Fitted" 2
"Lin. fit" 3 "Quad. fit") rows(3) holes(3) colfirst)
```

This legend is the same as the one in the previous two examples, but here we add the `holes(3)` option so a blank key is placed in the third position forcing the word *Fitted* in the fourth position at the top of the second column. *Uses allstatesdc.dta*



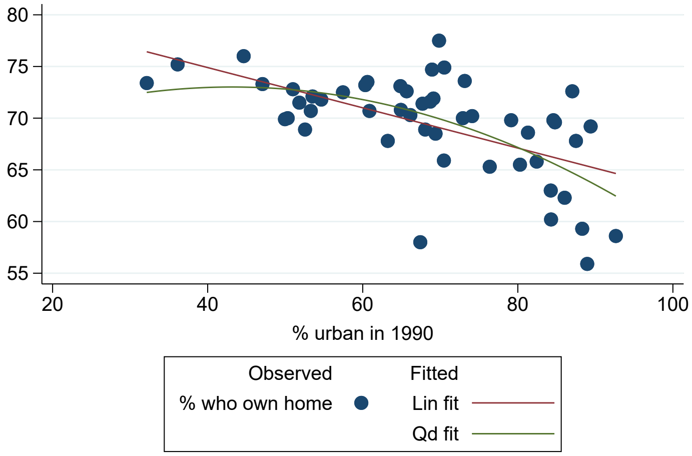
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Observed" 1 - " " - "Fitted"
2 "Lin fit" 3 "Qd fit") rows(3) colfirst)
```

Instead of using `holes()` as we did in the previous example to insert a blank key, we use `- " "` in the `order()` option, which pushes the word *Fitted* to the next column. *Uses allstatesdc.dta*



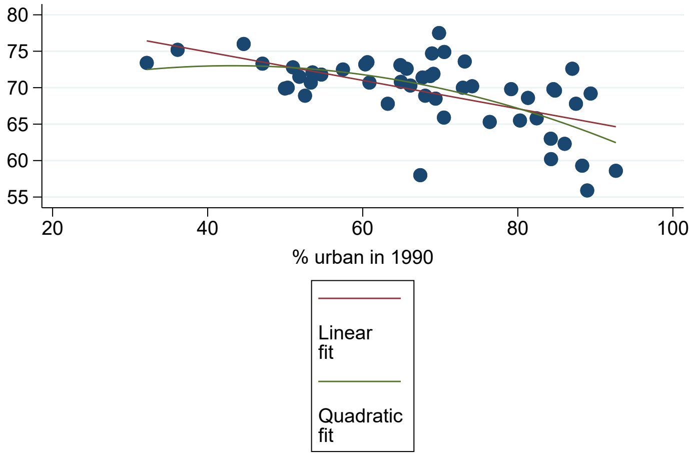
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(- "Observed" 1 - " " - "Fitted"
2 "Lin fit" 3 "Qd fit") rows(3) colfirst textfirst)
```

With the `textfirst` option, the text for the key appears first, followed by the symbol. *Uses allstatesdc.dta*



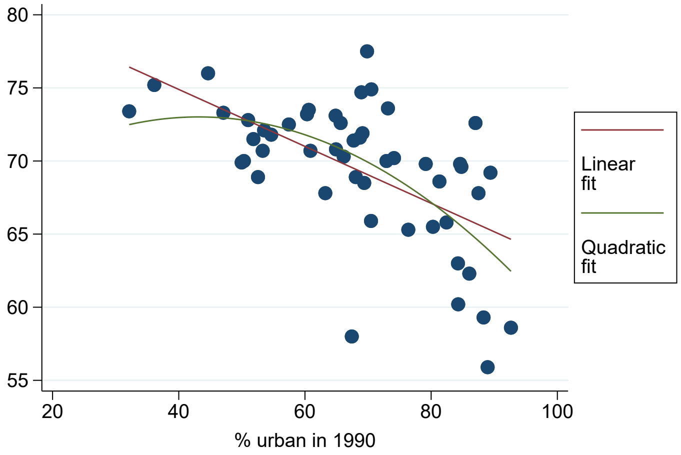
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(2 "Linear" "fit" 3 "Quadratic"
"fit") stack cols(1))
```

The `stack` option stacks the symbols above the labels. We use this option here to make a tall, narrow legend. *Uses allstatesdc.dta*



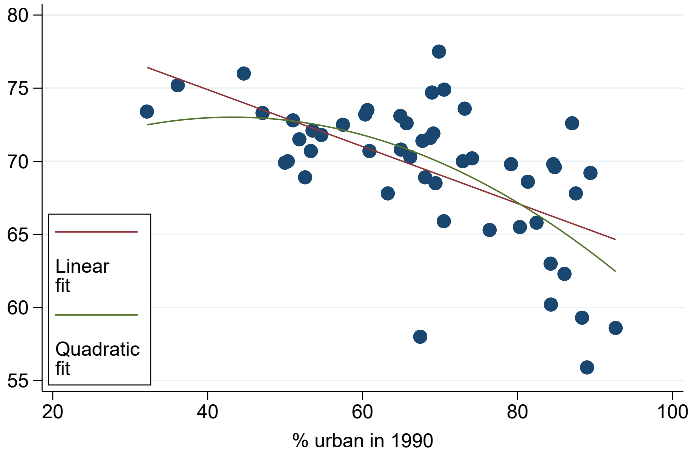
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(2 "Linear" "fit" 3 "Quadratic"
"fit") stack cols(1) position(3))
```

In this example, we use the `position()` option to move the narrow legend to the three o'clock position, right of the graph. *Uses allstatesdc.dta*



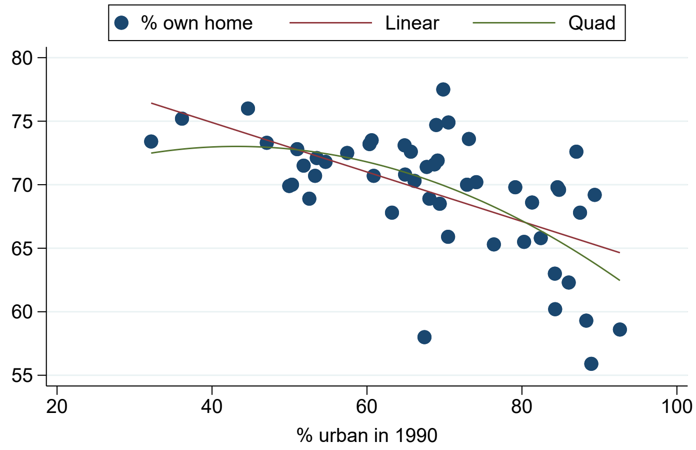
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(2 "Linear" "fit" 3 "Quadratic"
"fit") stack cols(1) ring(0) position(7))
```

We now use the `ring(0)` option to place the legend inside the plot area and use `position(7)` to put it in the bottom left corner, using the empty space in the plot for the legend. *Uses allstatesdc.dta*



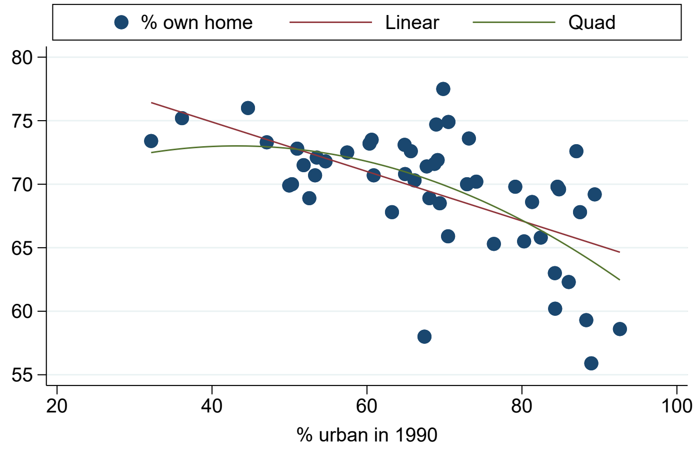
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(1 "% own home" 2 "Linear" 3
"Quad") rows(1) position(12))
```

Here we make the legend a thin row at the top of the graph by using the `rows(1)` and `position(12)` options. *Uses allstatesdc.dta*



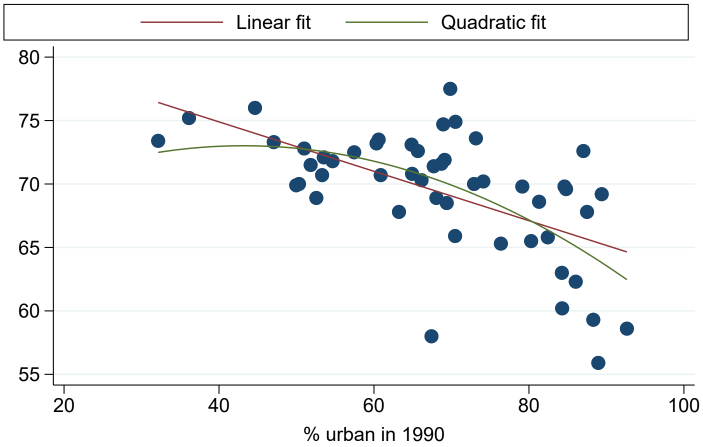
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(1 "% own home" 2 "Linear" 3
"Quad") rows(1) position(12) bexpand)
```

We can expand the width of the legend to the width of the plot area by using the `bexpand` (box expand) option. *Uses allstatesdc.dta*

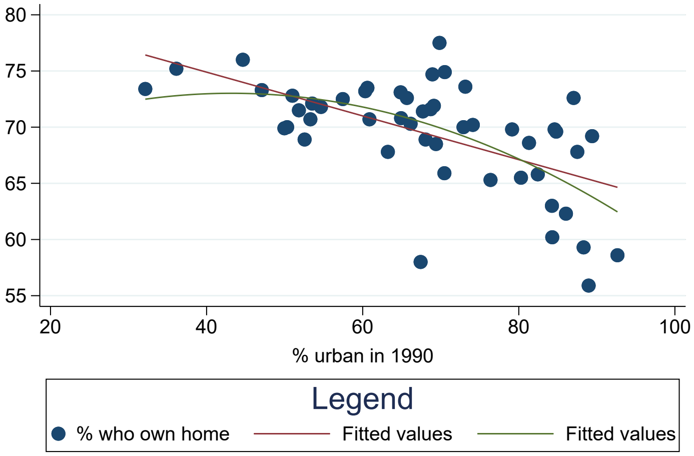


```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(order(2 "Linear fit" 3 "Quadratic
fit") rows(1) position(12) bexpand span)
```

The `span` option expands the legend to the entire width of the graph area (not just the plot area). *Uses allstatesdc.dta*



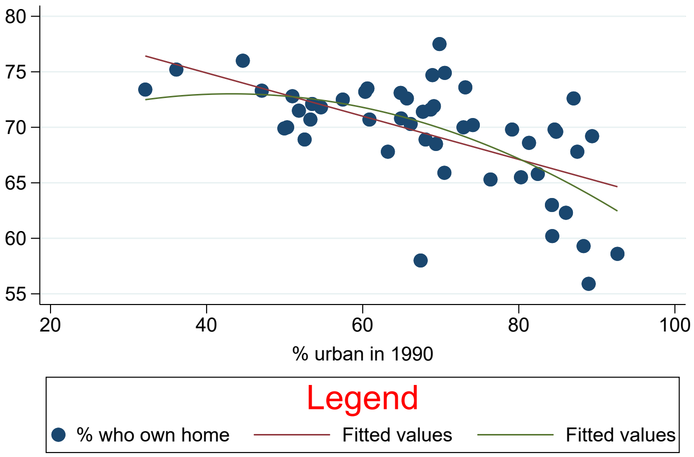
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend( rows(1) title("Legend"))
```



We can add a title, subtitle, note, or caption to the legend by using all the features described in Standard options : Titles (section 9.1). Here we add the `title()` option to include a title in the legend. A simple way to get a smaller title is to use the `subtitle()` option. *Uses allstatesdc.dta*

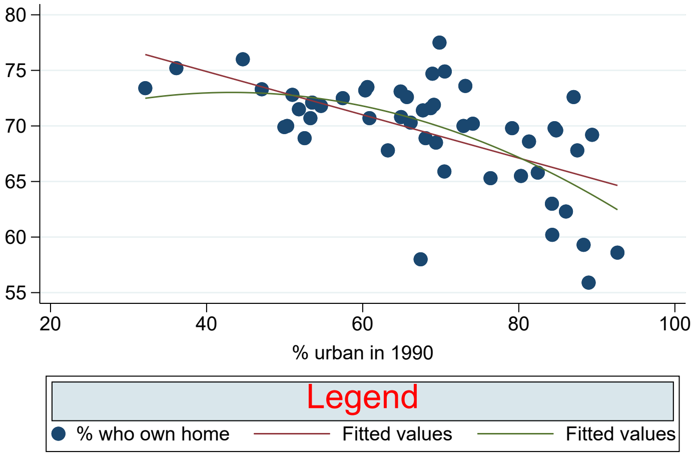
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend( rows(1) title("Legend", color(red)
size(huge)))
```

By using the `color()` and `size()` options, we make the legend title red and huge. *Uses allstatesdc.dta*



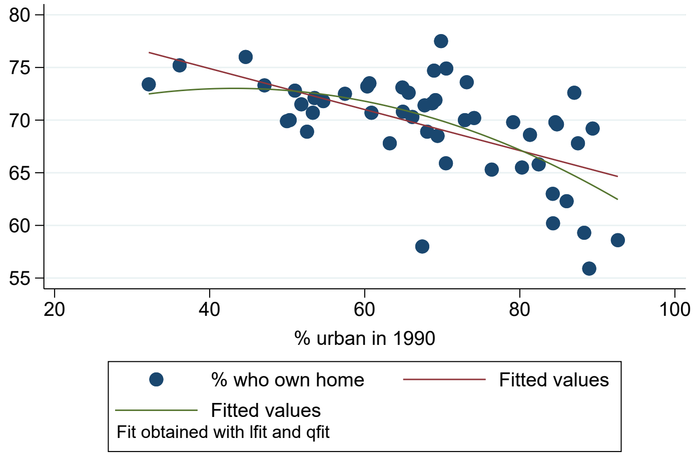
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(rows(1) title("Legend", color(red)
size(huge) box bexpand))
```

To emphasize the control we have, we put the title for the legend in a `box` and use `bexpand` to expand it the width of the legend. *Uses allstatesdc.dta*



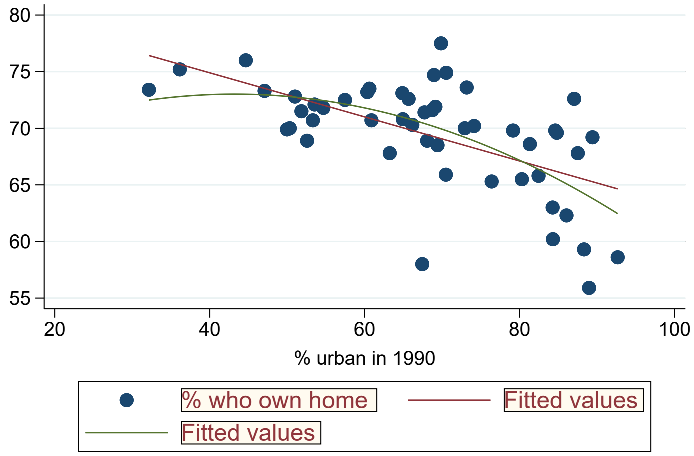
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(note("Fit obtained with lfit and
qfit"))
```

Here we use the `note()` option to show that a note can be added to the legend. *Uses allstatesdc.dta*



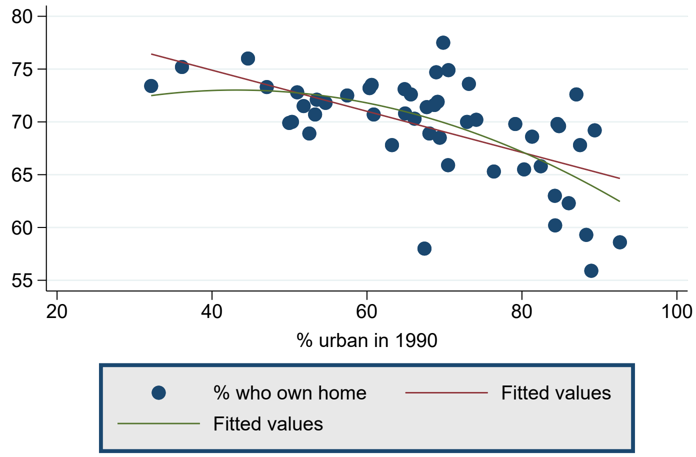
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(size(large) color(maroon)
fcolor(eggshell) box)
```

We can also control the display of the labels for the keys with the `legend()` option. Here we request that those labels be large and maroon and be displayed with an eggshell background surrounded by a box. *Uses allstatesdc.dta*



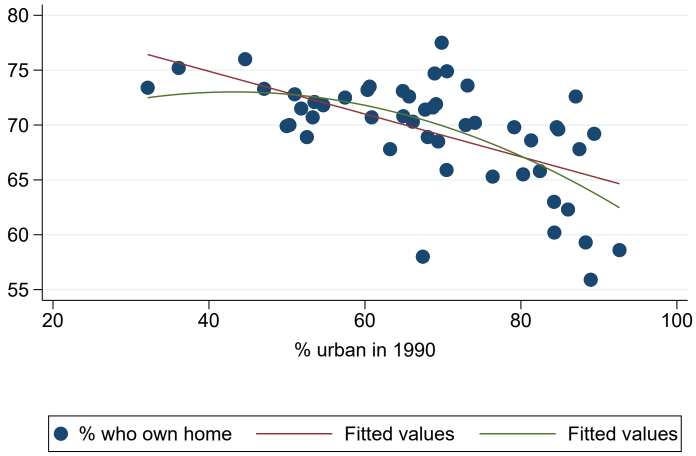
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(region(fcolor(dimgray) lcolor(navy)
lwidth(thick) margin(medium)))
```

The `region()` option controls the overall box in which the legend is placed. Here we specify the fill color dim gray and the line thick and navy with a medium-sized margin between the text and the box. *Uses allstatesdc.dta*



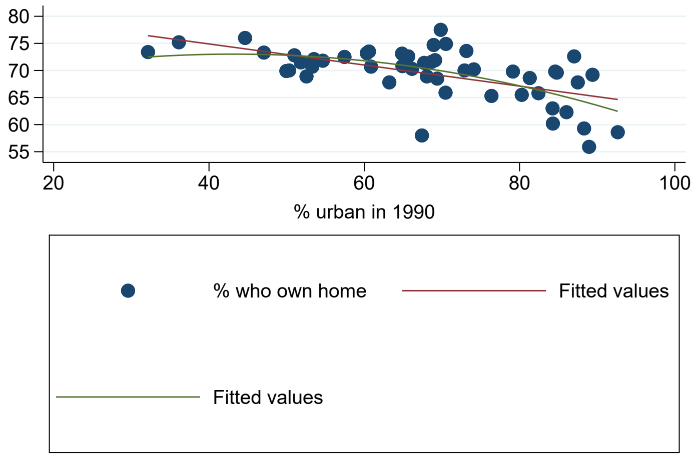
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(rows(1) bmargin(t=10))
```

The `bmargin()` option adjusts the margin around the box of the legend. Here we make the margin 10 at the top, which increases the gap between the legend and the title of the x axis. *Uses allstatesdc.dta*

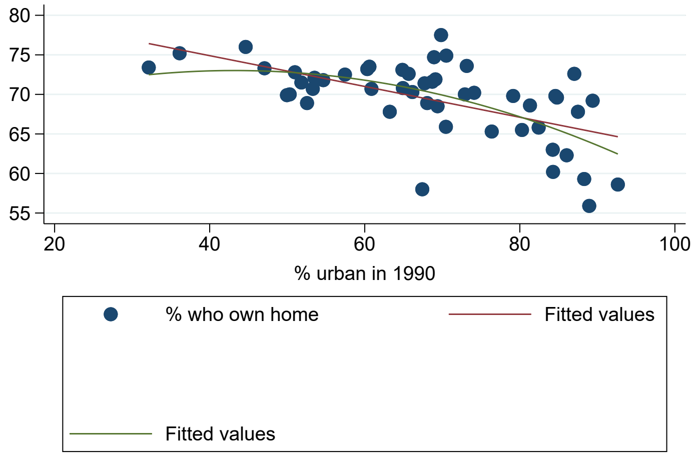


```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(symxsize(30) symysize(20))
```

We control the width allocated to symbols with the `symxsize()` option and the height with the `symysize()` option. *Uses allstatesdc.dta*



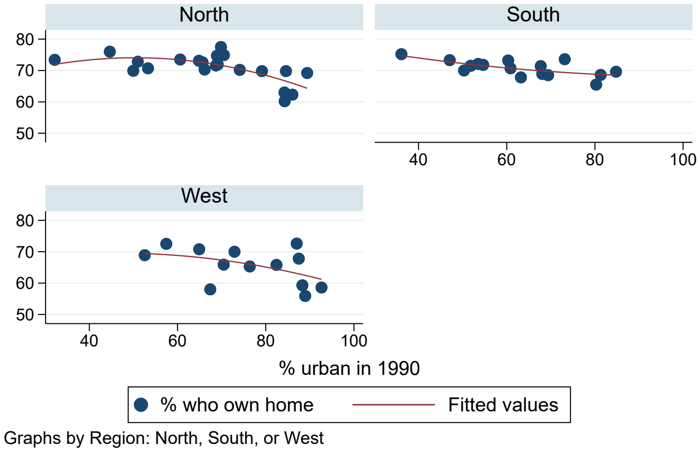
```
twoway (scatter ownhome urban) (lfit ownhome urban) (qfit
ownhome urban), legend(colgap(25) rowgap(20))
```



Here we control the space between columns of the legend with the `colgap()` option and the space between the rows with the `rowgap()` option. The `rowgap()` option does not affect the border between the top row and the box or the border between the bottom row and the box. *Uses allstatesdc.dta*

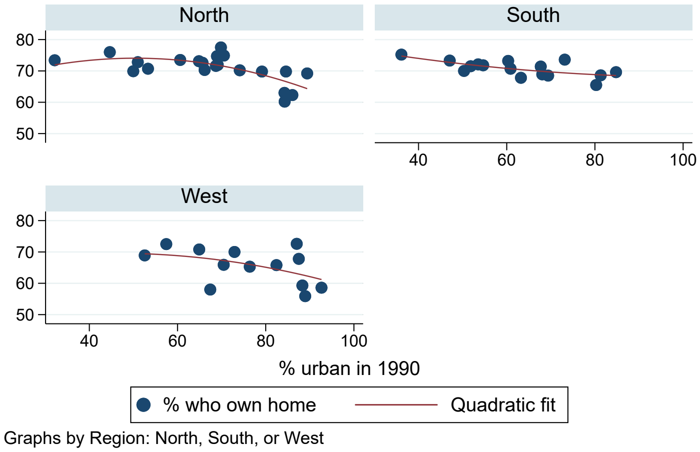
```
twoway (scatter ownhome urban) (qfit ownhome urban), by(nsw)
```

Consider this graph showing two overlaid scatterplots shown separately by the location of the state. We will now explore how to modify the legend for this kind of graph. *Uses allstatesdc.dta*



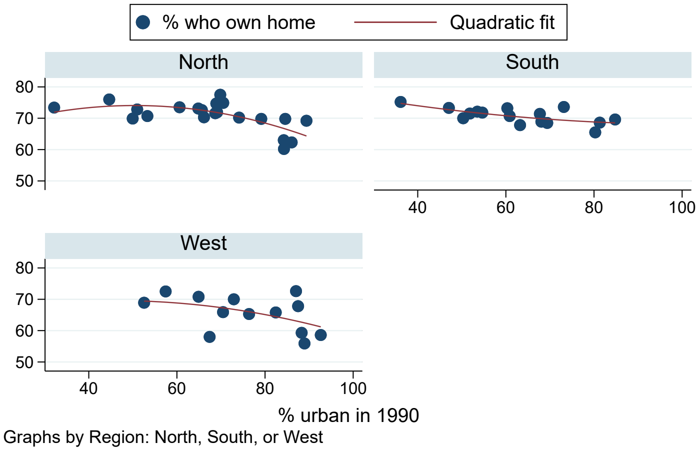
```
twoway (scatter ownhome urban) (qfit ownhome urban), by(nsw)
legend(position(12) label(2 "Quadratic fit"))
```

Here we add a `legend()` option, but the `position()` option has no effect because the position of the legend did not change. *Uses allstatesdc.dta*



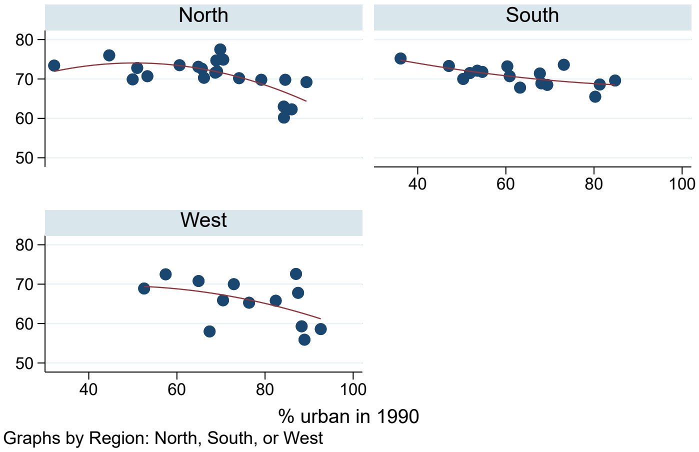
```
twoway (scatter ownhome urban) (qfit ownhome urban), by(nsw,
legend(position(12))) legend(label(2 "Quadratic fit"))
```

The previous command did not change the position of the legend because options for positioning the legend must be placed within the `by()` option. Here we place the `legend(position())` option within the `by()` option, and the legend is now placed above the graph. *Uses allstatesdc.dta*



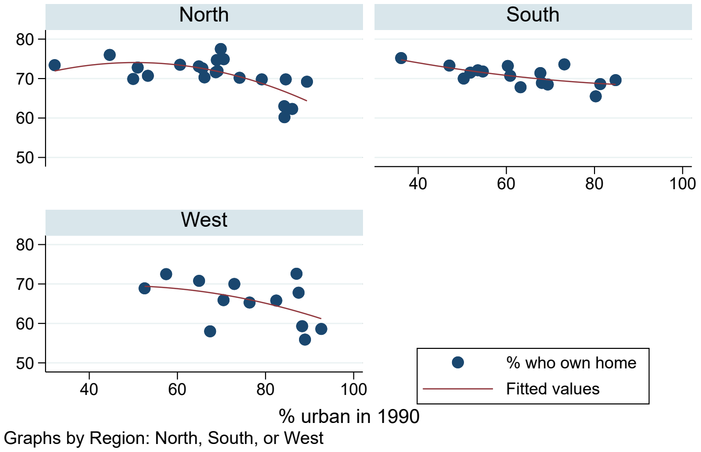
```
twoway (scatter ownhome urban) (qfit ownhome urban), by(nsw,
legend(off))
```

Likewise, if we wish to turn the legend off, we must place `legend(off)` within the `by()` option. *Uses allstatesdc.dta*



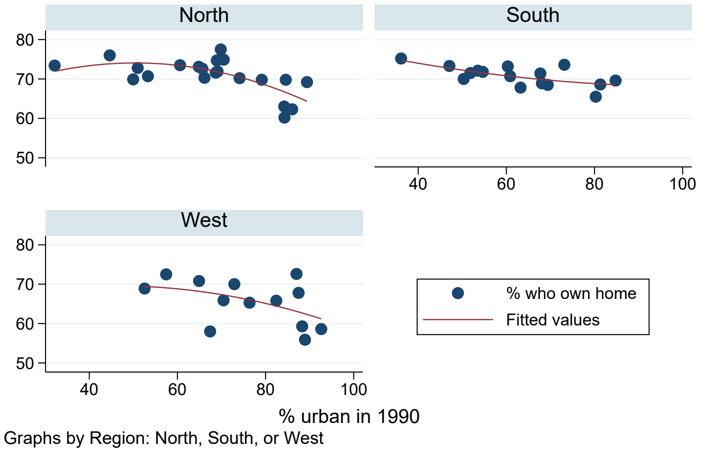
```
twoway (scatter ownhome urban) (qfit ownhome urban), by(nsw,
legend(at(4))) legend(cols(1))
```

Here we place the legend in the fourth hole by using the `at(4)` option within the `by()` option. To display the legend in one column, we use the `legend(cols(1))` option outside the `by()` option because this option does not effect the position of the legend. *Uses allstatesdc.dta*



```
twoway (scatter ownhome urban) (qfit ownhome urban), by(nsw,
legend(position(center) at(4))) legend(cols(1))
```

In this example, we add the `position(center)` option within the `by()` option to make the legend appear in the center of the fourth position. *Uses allstatesdc.dta*

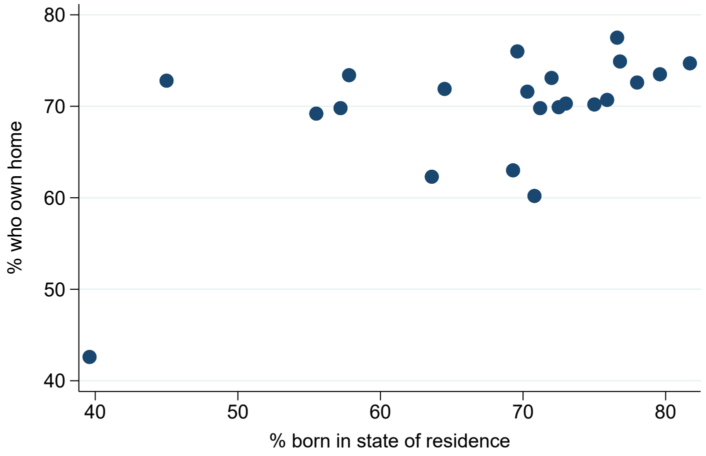


8.10 Adding text to markers and positions

This section provides more details about the `text()` option for adding text to a graph. Although we can use added text in a wide variety of situations, this section will focus on how we can use it to label points and lines and to add descriptive text to our graph. For more information about the `text()` option, see `help added_text_options`. To learn more about how you can customize the text, see Options : Textboxes (section 8.11).

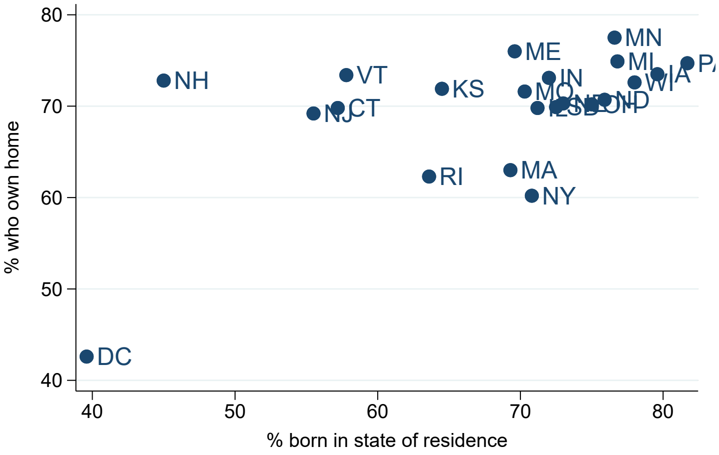
```
twoway scatter ownhome borninstate
```

In this scatterplot, one point appears to be an outlier. Because it is not labeled, we cannot tell from which state it originates. *Uses allstatesn.dta*



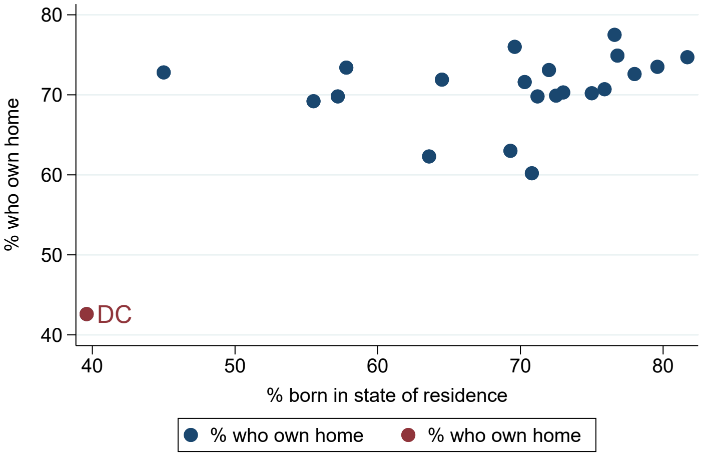
```
scatter ownhome borninstate, mlabel(stateab)
```

We use the `mlabel(stateab)` to label all points, which helps us see that the outlying point comes from Washington, DC. However, this plot is rather cluttered by all the labels. *Uses allstatesn.dta*



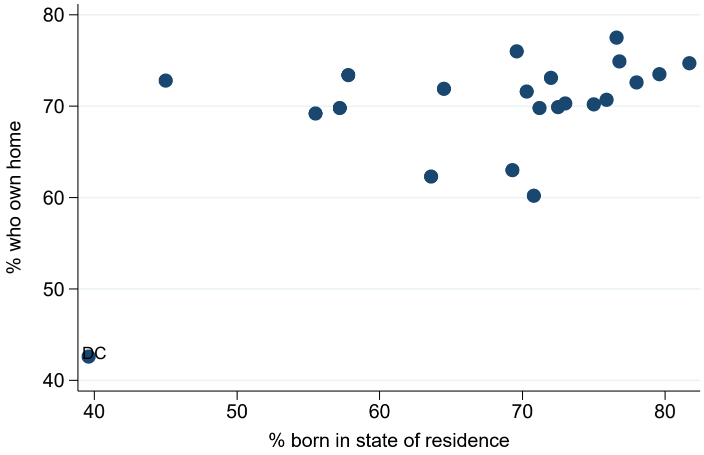
```
twoway (scatter ownhome borninstate) (scatter ownhome
borninstate if stateab == "DC", mlabel(stateab))
```

Here we repeat a second scatterplot just to label DC, but this is a bit cumbersome. *Uses allstatesn.dta*



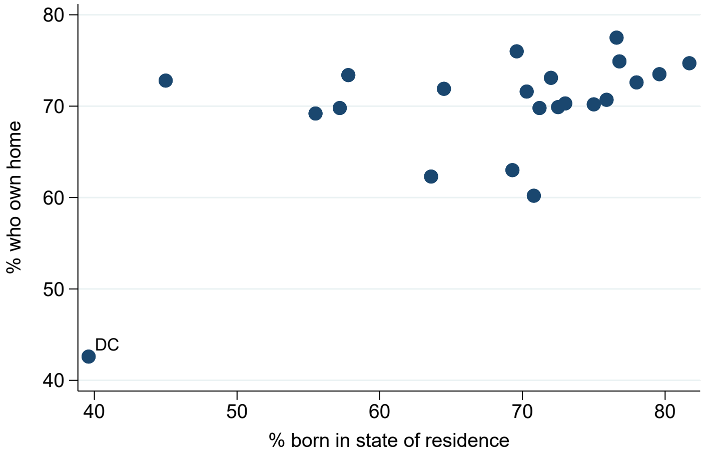
```
twoway scatter ownhome borninstate, text(43 40 "DC")
```

Instead, we use the `text()` option to add text to our graph. Looking at the values of `ownhome` and `borninstate` for DC, we see that their values are about 43 and 40, respectively. We use these as coordinates to label the point, but the `text()` option places the label at the center of the specified y x coordinate, sitting right over the point. *Uses allstatesn.dta*



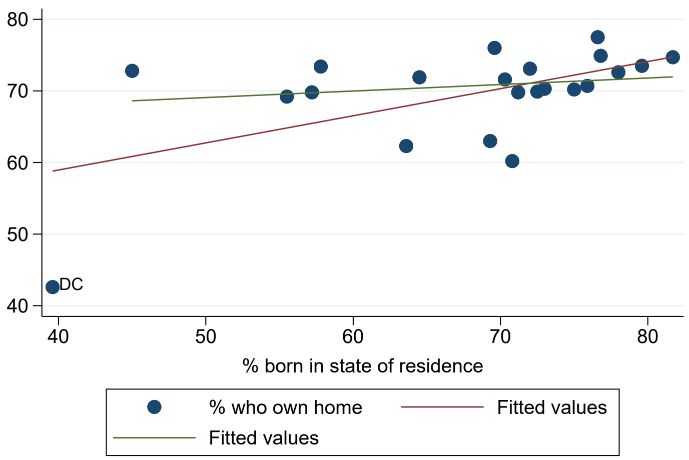
```
twoway scatter ownhome borninstate, text(43 40 "DC",
placement(ne))
```

Adding the `placement(ne)` option within the `text()` option places the label above and to the right (northeast) of the point. Available placements include `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw`, and `c` (center); see Styles : Compassdir (section 10.4) for more details. *Uses allstatesn.dta*

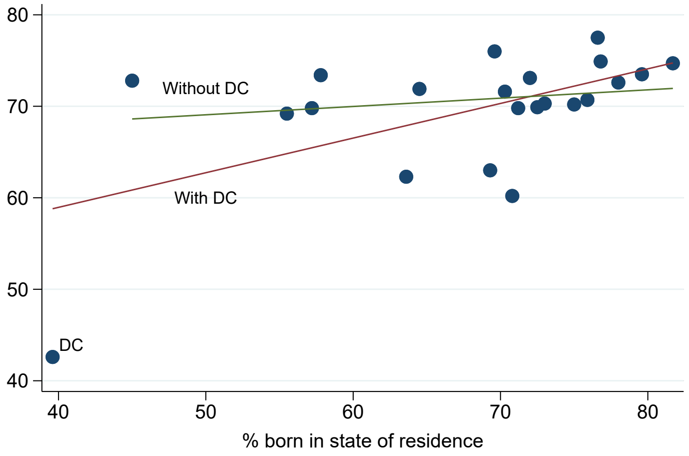


```
twoway (scatter ownhome borninstate, text(43 40 "DC",
placement(e))) (lfit ownhome borninstate) (lfit ownhome
borninstate if stateab !="DC")
```

Consider this scatterplot showing a linear fit between the two variables: one including Washington, DC, and one omitting Washington, DC. See the next graph, which uses the `text()` option to label the graph instead of the legend. *Uses allstatesn.dta*



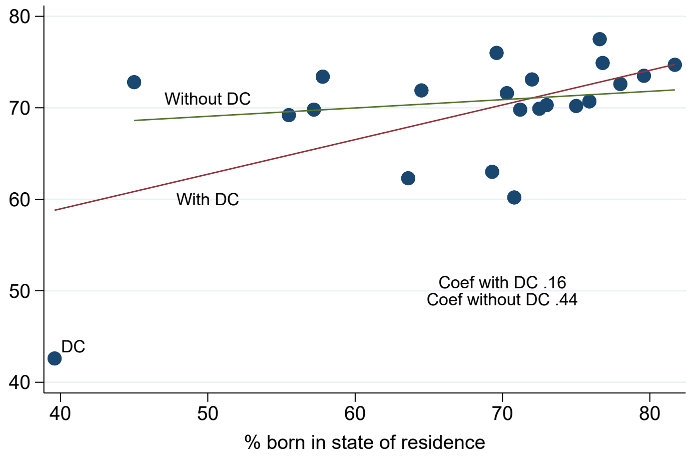
```
twoway (scatter ownhome borninstate, text(43 40 "DC",
placement(ne))) (lfit ownhome borninstate) (lfit ownhome
borninstate if stateab !="DC", text(72 50 "Without DC")
text(60 50 "With DC")), legend(off)
```



This graph turns the legend off and uses the `text()` option to label each regression line to indicate which regression line includes DC and which one excludes DC. *Uses allstatesn.dta*

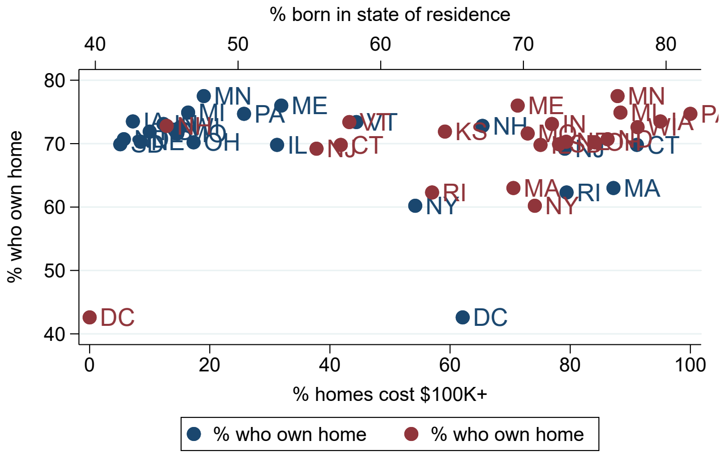
```
twoway (scatter ownhome borninstate, text(43 40 "DC",
placement(ne))) (lfit ownhome borninstate) (lfit ownhome
borninstate if stateab !="DC", text(71 50 "Without DC")
text(60 50 "With DC") text(50 70 "Coef with DC .16" "Coef
without DC .44")), legend(off)
```

This graph adds explanatory text, showing the regression coefficient with and without DC. *Uses allstatesn.dta*



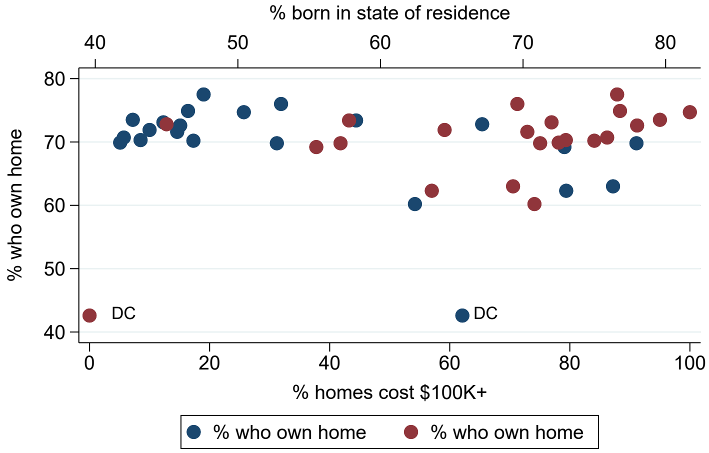
```
twoway (scatter ownhome propval100, xaxis(1) mlabel(stateab))
(scatter ownhome borninstate, xaxis(2) mlabel(stateab))
```

Consider this graph in which we overlay two scatterplots. We place `propval100` on the first x axis and `borninstate` on the second x axis. *Uses allstatesn.dta*



```
twoway (scatter ownhome propval100, xaxis(1)) (scatter
ownhome borninstate, xaxis(2)), text(43 66 "DC") text(43 42
"DC", xaxis(2))
```

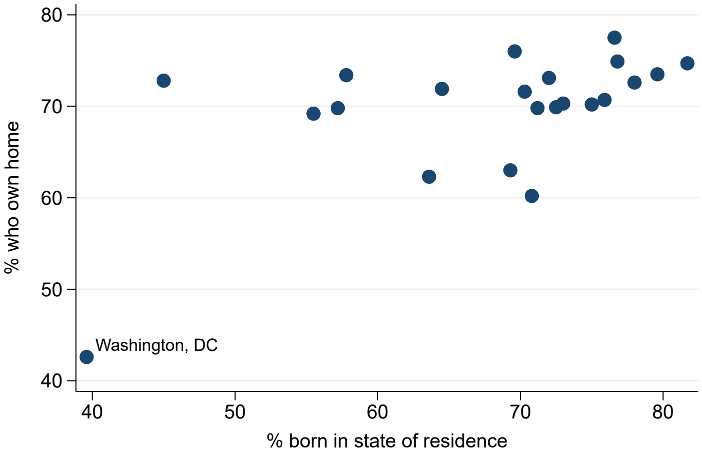
Rather than labeling all the points, we can label just the point for DC. We must be careful because we have two different x axes. The first `text()` option uses the first x axis, so no special option is required. The second `text()` option uses the second x axis, so we must specify the `xaxis(2)` option. *Uses allstatesn.dta*


% born in state of residence
40
50
60
70
80
80
70
60
50
40
% who own home
DC
DC
0
20
40
60
80
100
% homes cost $100K+
● % who own home
● % who own home

8.11 Options for text and textboxes

This section describes more options for modifying textbox elements: titles, captions, notes, added text, and the legend. Technically, all text in a graph is displayed within a textbox. We can modify the box's attributes, such as its size and color, the margin around the box, and the outline; and we can modify the attributes of the text within the box, such as its size, color, justification, and margin. These examples sometimes include the `box` option to see how both the textbox and its text are being displayed. This helps us to see if we should modify the attributes of the box containing the text or the text within the box. For more information, see `help textbox_options` and Options : Adding text (section 8.10). This section begins by showing examples illustrating how to control the placement, size, color, and orientation of text.

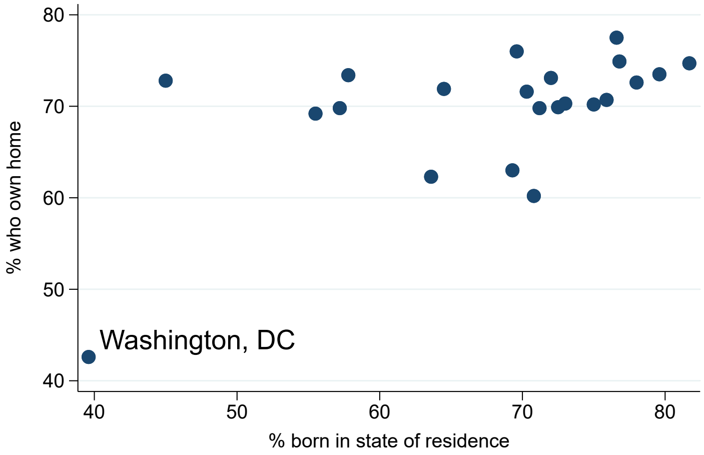
```
twoway scatter ownhome borninstate, text(43 40 "Washington,
DC", placement(ne))
```



Consider this scatterplot, which has a dramatic outlying point. We have used the `text()` option to label that point, but, perhaps, we might want to control the size of the text for this label. See the next example for an illustration of how to do this. *Uses allstatesn.dta*

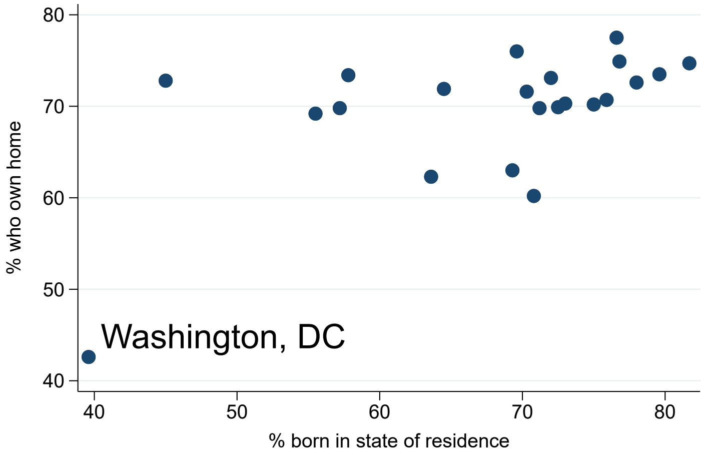
```
twoway scatter ownhome borninstate, text(43 40 "Washington,
DC", placement(ne) size(vlarge))
```

We can alter the size of the text by using the `size()` option. Here we make the text very large. Available sizes include `zero`, `miniscule`, `quarter_tiny`, `third_tiny`, `half_tiny`, `tiny`, `vsmall`, `small`, `medsmall`, `medium`, `medlarge`, `large`, `vlarge`, `huge`, and `vhuge`; see **Styles : Textsize** (section 10.12) for more details (including sizing using points, inches, and centimeters). *Uses allstatesn.dta*



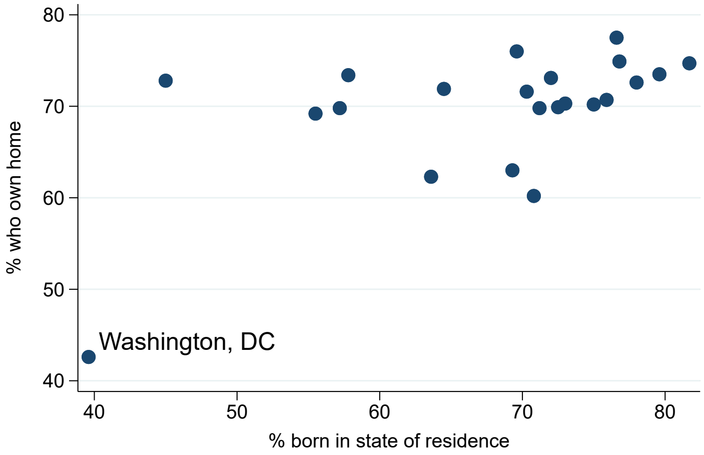
```
twoway scatter ownhome borninstate, text(43 40 "Washington,
DC", placement(ne) size(*2))
```

This example specifies a multiplier for the size of the text, making the text display twice as large as the original text. Had we specified `size(*.5)`, the text would have been displayed to half its original size. For more details, see **Styles : Textsize** (section 10.12) (including sizing using points, inches, and centimeters). *Uses allstatesn.dta*



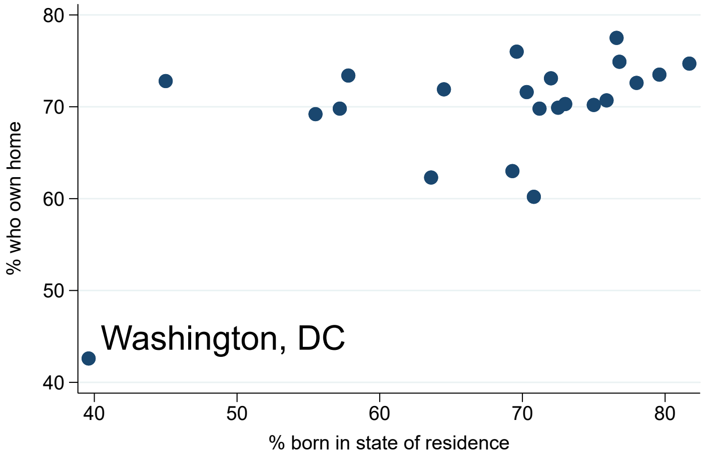
```
twoway scatter ownhome borninstate, text(43 40 "Washington,
DC", placement(ne) size(5rs))
```

This shows a third way we can control the size of text, specifying `size(5rs)`. The size of the text is set as 5% of the size of the graph (the minimum of the width and height). This graph is 3 by 2 inches, so the text will be 5% of 2, or 0.1 inch. For more details, see **Styles : Textsize** (section 10.12) (including sizing using points, inches, and centimeters). *Uses allstatesn.dta*



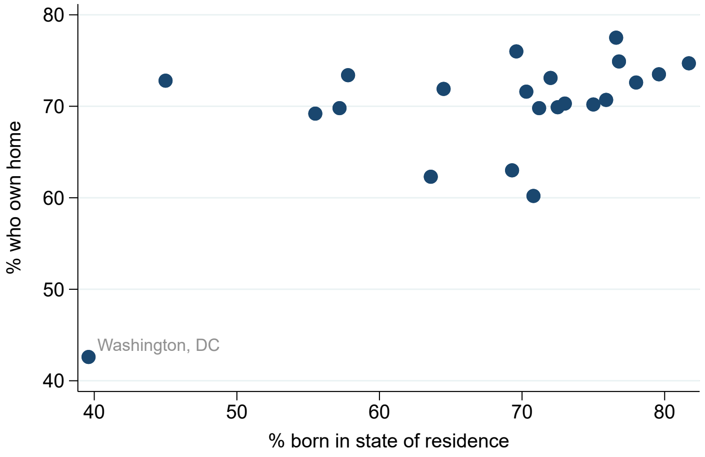
```
twoway scatter ownhome borninstate, text(43 40 "Washington,
DC", placement(ne) size(10pt))
```

You can also specify the size of text using points, inches, or centimeters. The text in this example is displayed as 10 points. Specifying `size(.5in)` would make the text half an inch, and `size(1cm)` would size the text as 1 centimeter. For more details, see **Styles : Textsize** (section 10.12). Also see section 9.3.1 regarding resizing graphs when using absolute units. *Uses allstatesn.dta*

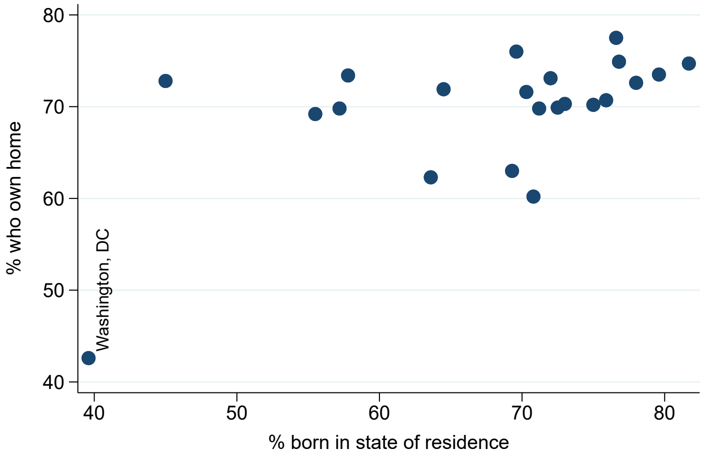


```
twoway scatter ownhome borninstate, text(43 40 "Washington,
DC", placement(ne) color(gs9))
```

We can alter the color of the text by using the `color()` option. Here we make the text a middle-level gray. See **Styles : Colors** (section 10.2) for other colors you could use. *Uses allstatesn.dta*



```
twoway scatter ownhome borninstate, text(43 40 "Washington,
DC", placement(ne) orientation(vertical))
```

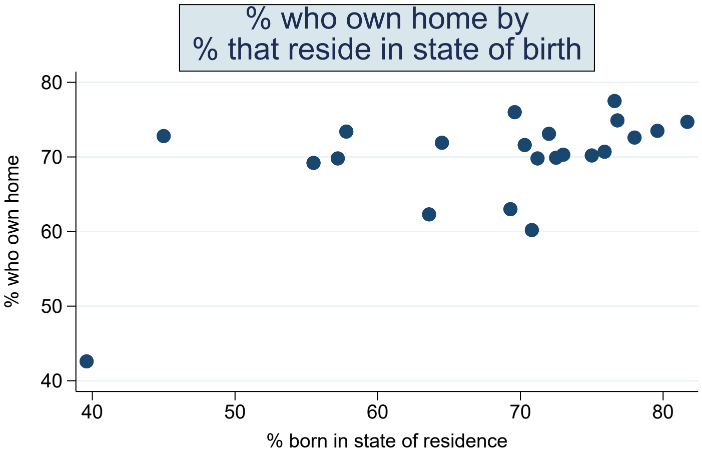


The `orientation()` option changes the direction of the text. Available orientations include `horizontal` for 0 degrees, `vertical` for 90 degrees, `rhorizontal` for 180 degrees, and `rvertical` for 270 degrees. See Styles : Orientation (section 10.10) for more details. *Uses allstatesn.dta*

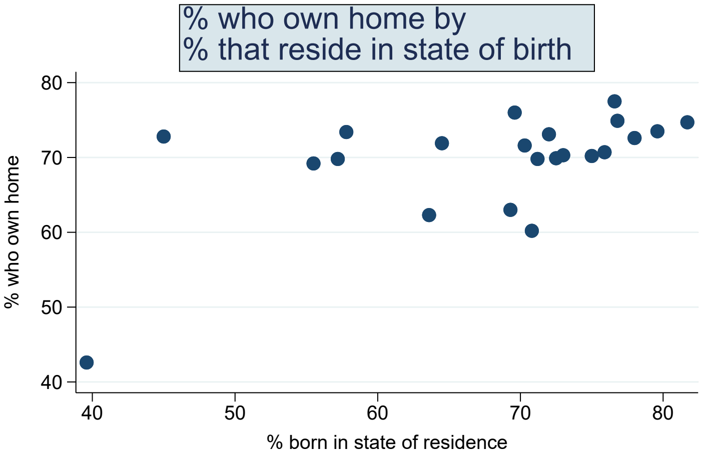
This next set of examples considers options for justifying text within a box, sizing the box, and creating margins around the box. This is followed by options that control margins within the textbox.

```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box)
```

Consider this example where we place a title on our graph. To help show how the options work, we put a `box` around the title. *Uses allstatesn.dta*



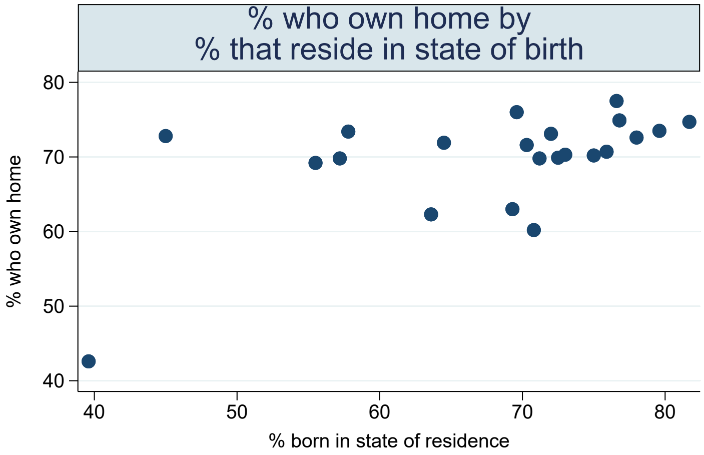
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box
justification(left))
```



In this example, we left justify the text by using the `justification()` option. The title is justified within the textbox, not with respect to the entire graph area. *Uses allstatesn.dta*

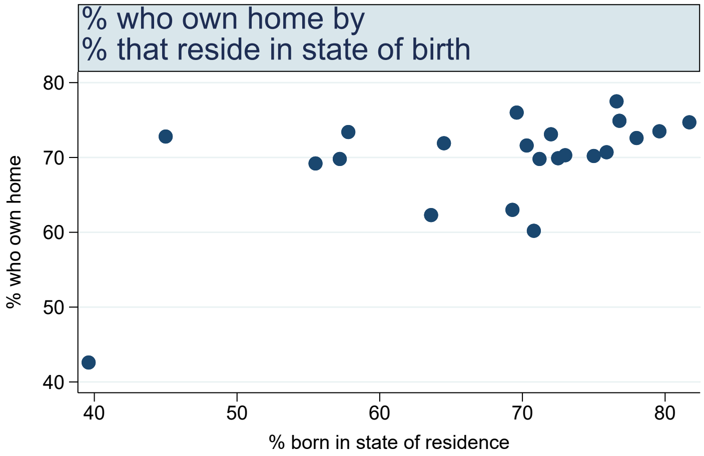
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box bexpand)
```

By using the `bexpand` (box expand) option, the textbox containing the title expands to fill the width of the plot area. *Uses allstatesn.dta*



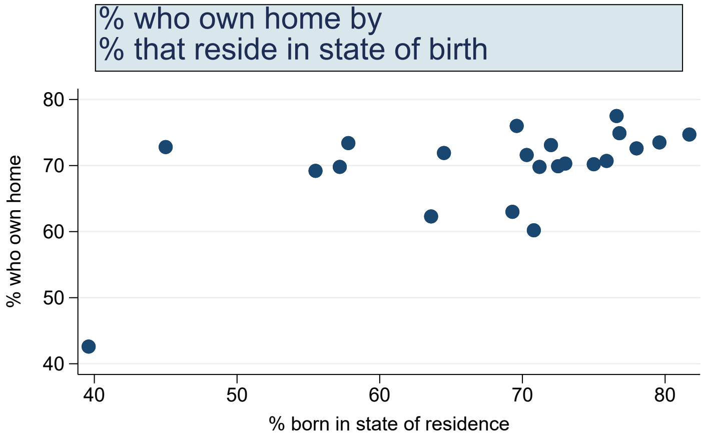
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box bexpand
justification(left))
```

With the box expanded, the `justification(left)` option now makes the title flush left with the plot area. *Uses allstatesn.dta*



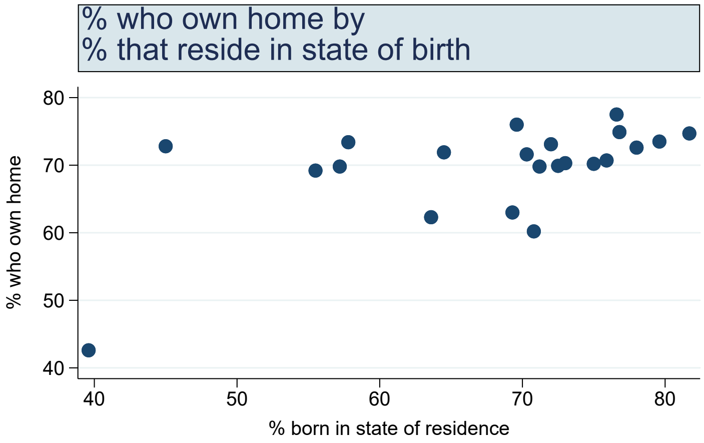
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box bexpand
justification(left) bmargin(medium))
```

We can change the size of the margin around the outside of the box by using the `bmargin(medium)` (box margin) option. Here we make the margin medium-sized at all four edges: left, right, top, and bottom. *Uses allstatesn.dta*



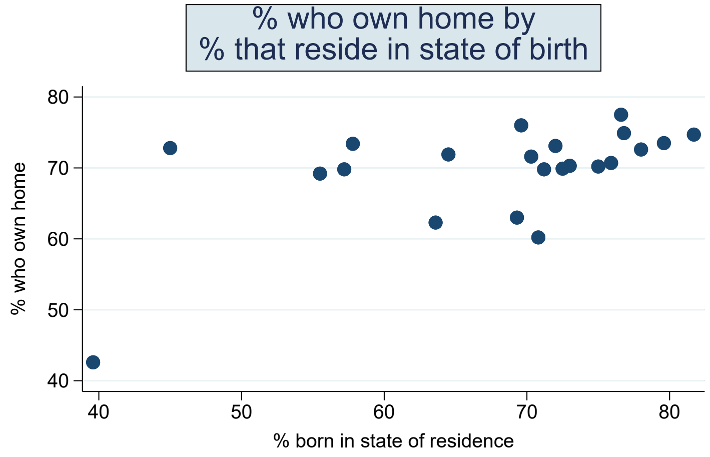
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box bexpand
justification(left) bmargin(0 0 3 3))
```

The margin around the title is 0 for the left and right and 3 for the top and bottom. The order of the margins is `bmargin(#left # right #top #bottom)`. *Uses allstatesn.dta*

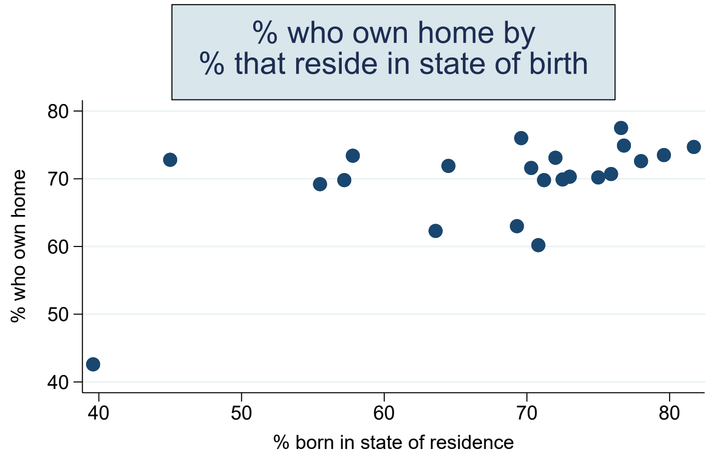


```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box bmargin(b=3))
```

To make only the bottom margin 3, we specify `bmargin(b=3)`, where `b=3` means to change the bottom margin to 3. The left, right, top, and bottom margins can be changed individually using `l=`, `r=`, `t=`, and `b=`, respectively. *Uses allstatesn.dta*



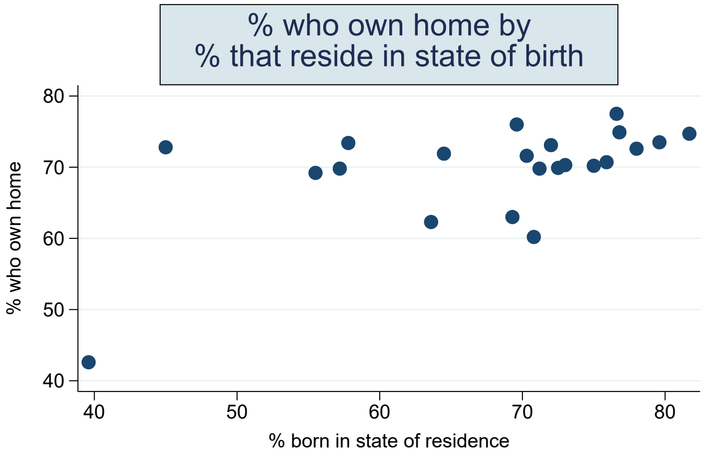
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box margin(medium))
```



We can expand the margin between the text and the box by using the `margin()` option. Note the difference between this and the `bmargin()` option (illustrated previously) is the increased margin around the text inside the box. *Uses allstatesn.dta*

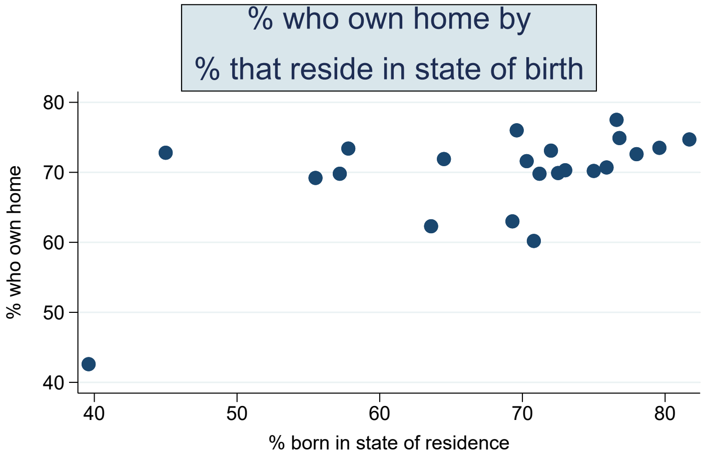
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box margin(5 5 2 2))
```

As with the `bmargin()` option, we can more precisely modify the margin around the text. Here we use the `margin()` option to make the size of the margin 5, 5, 2, and 2 for the left, right, top, and bottom, respectively. *Uses allstatesn.dta*



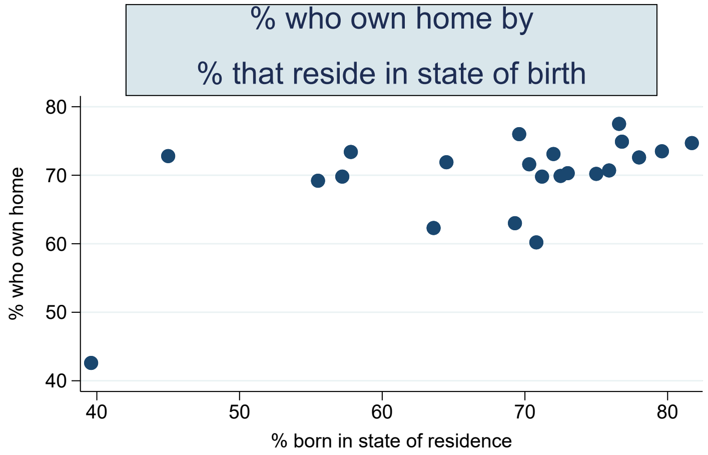
```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box linegap(4))
```

We can change the gap between the lines with the `linegap()` option. Here we make the gap larger than it would normally be. See Styles : Margins (section 10.8) for more details. *Uses allstatesn.dta*



```
twoway (scatter ownhome borninstate), title("% who own home
by" "% that reside in state of birth", box margin(l=.25in
r=.25in) linegap(.1in))
```

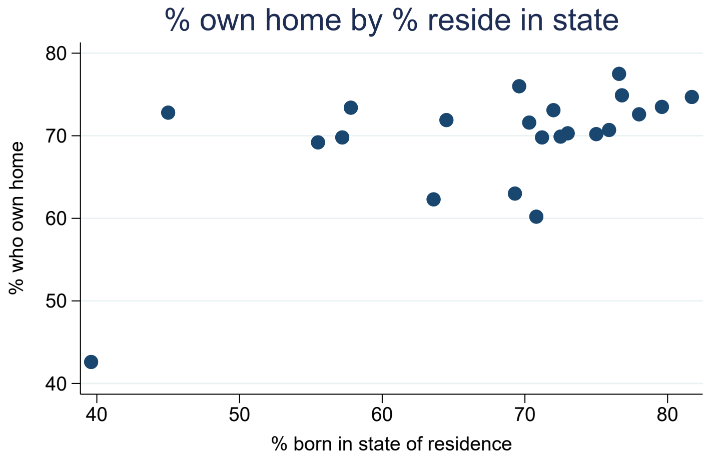
We can also specify the size of margins and line gaps using points, inches, or centimeters. In this example, we use the `margin(l=.25in r=.25in)`) and `linegap(.1in)` options to make the left and right margins a quarter of an inch and the line gap a tenth of an inch. See Styles : Margins (section 10.8) for more details. *Uses allstatesn.dta*



Let’s now consider options that control the color of the textbox and the characteristics of the outline of the box (including the color, thickness, and pattern).

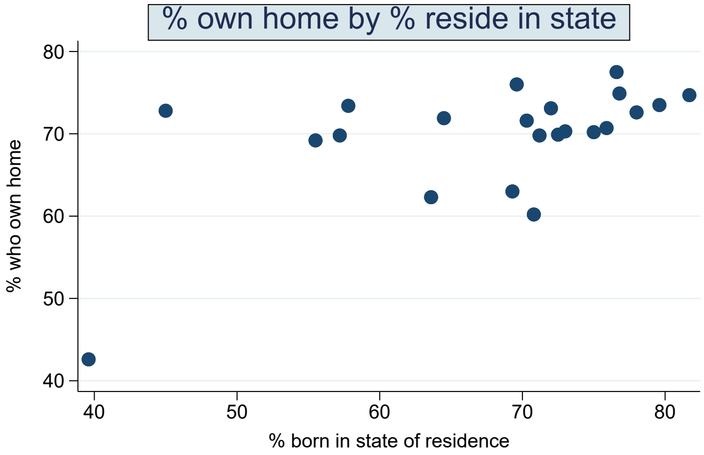
```
twoway (scatter ownhome borninstate), title("% own home by %
reside in state")
```

Consider this graph with a title at the top. *Uses allstatesn.dta*



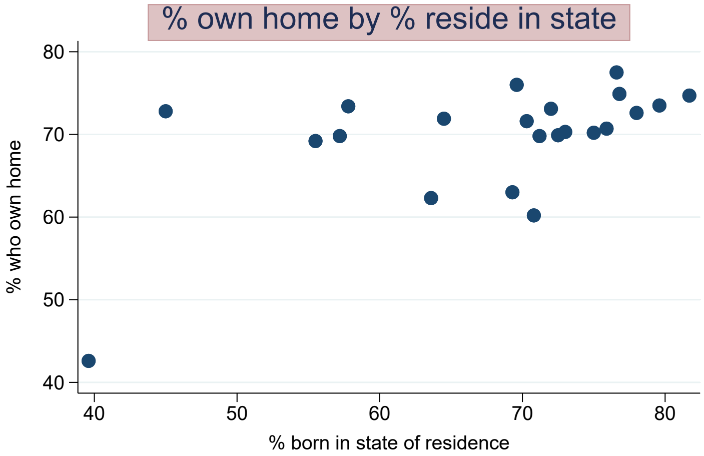
```
twoway (scatter ownhome borninstate), title("% own home by %
reside in state", box)
```

We add the `box` option now for aesthetic purposes. *Uses allstatesn.dta*



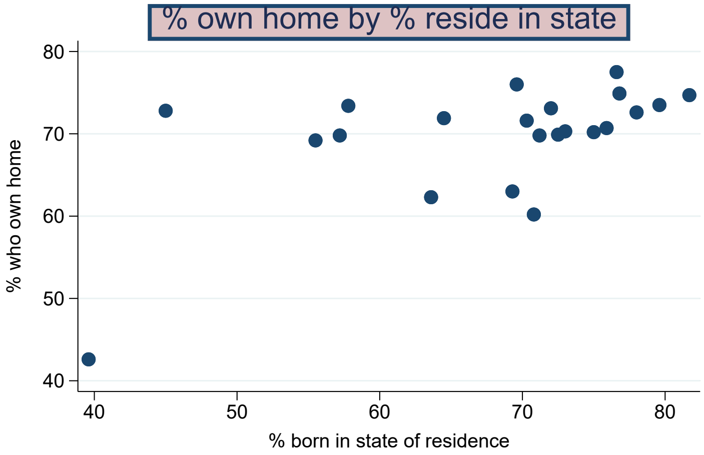
```
twoway (scatter ownhome borninstate), title("% own home by %
reside in state", box bcolor(maroon%30))
```

This example specified the option `bcolor(maroon%30)`, which changes the box color to `maroon` with 30% opacity.
Uses allstatesn.dta



```
twoway (scatter ownhome borninstate), title("% own home by %
reside in state", box fcolor(maroon%30) lcolor(navy)
lwidth(thick))
```

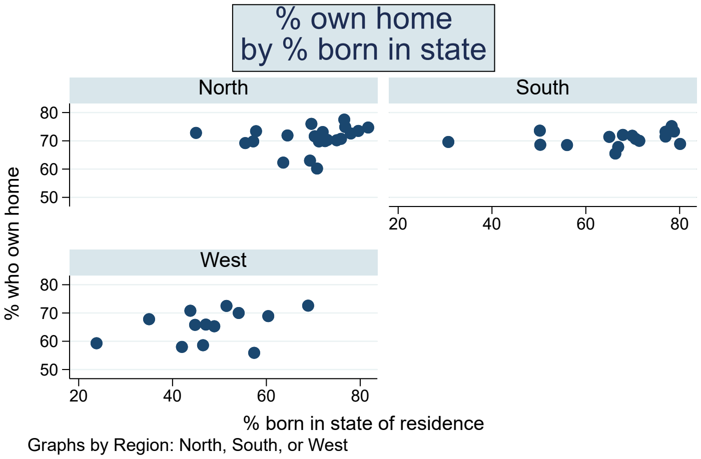
We can change the box fill color with the `fcolor()` option, the color of the line around the box with `lcolor()`, and the width of the surrounding box line with `lwidth()`. This example makes the box fill color `maroon` with 30% opacity and the box line color as `navy`. See Styles : Colors (section 10.2) for other values available with the `fcolor()` and `lcolor()` options and Styles : Linewidth (section 10.7) for other values available for `lwidth()`.
Uses allstatesn.dta



Let's now use the `allstates` file and consider some examples that use the `by()` option to display multiple graphs broken down by the location of the state. We will look at options for placing and aligning text in graphs that use the `by()` option.

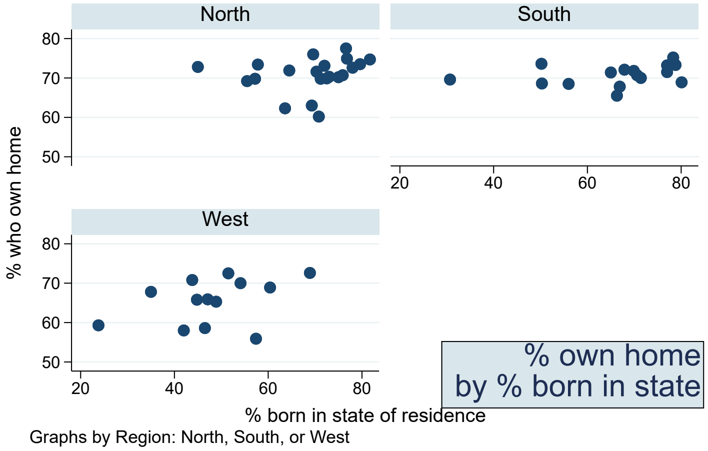
```
scatter ownhome borninstate, by(nsw, title("% own home" "by %
born in state", box))
```

Consider this graph in which we use the `by()` option to show this scatterplot separately for states in the North, South, and West. We include the `box` option only to show the outline of the textbox, not for aesthetics. *Uses allstates.dta*



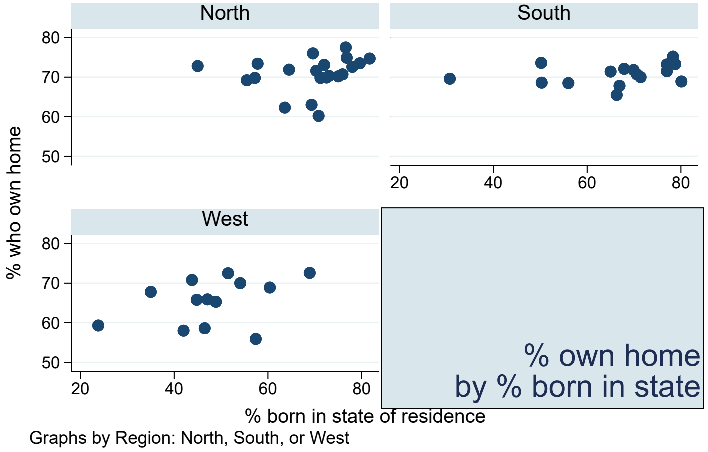
```
scatter ownhome borninstate, by(nsw, title("% own home" "by %
born in state", box ring(0) position(5)))
```

Let's put the title in the open hole in the right corner of the graph by using the `ring(0)` and `position(5)` options. *Uses allstates.dta*



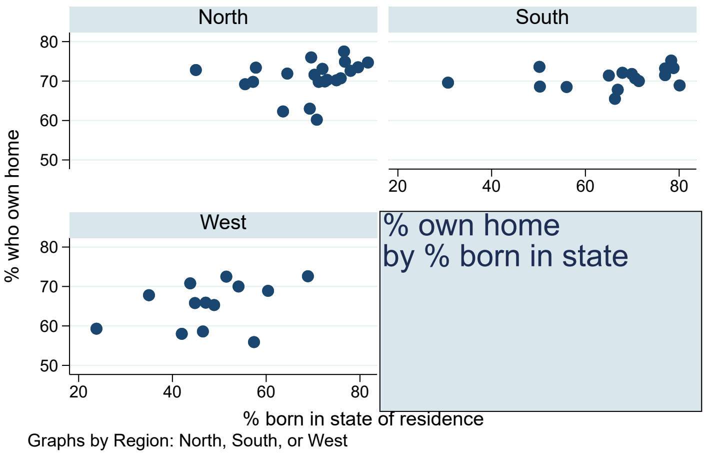
```
scatter ownhome borninstate, by(nsw, title("% own home" "by %
born in state", ring(0) position(5) box width(65)
height(40)))
```

We can make the area for the textbox bigger by using the `width()` and `height()` options. Here we change the value to make the box approximately as tall as the graph for the West and as wide as the graph for the South. *Uses allstates.dta*



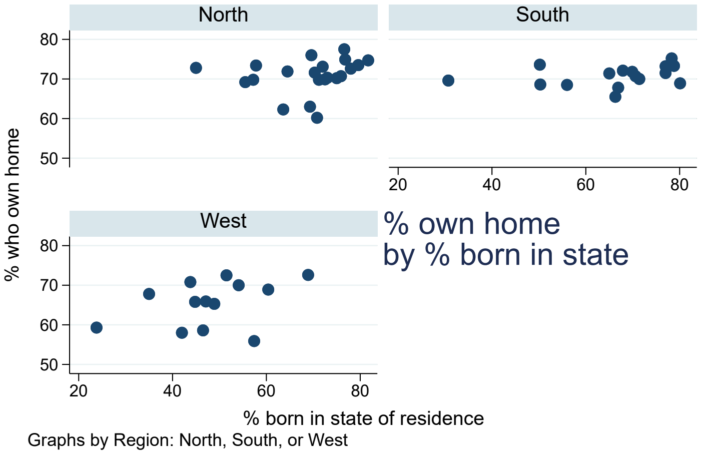
```
scatter ownhome borninstate, by(nsw, title("% own home" "by %
born in state", ring(0) position(5) box width(65) height(40)
justification(left) alignment(top)))
```

In this example, we left justify the text and align it with the top by using the `justification(left)` and `alignment(top)` options. These options make the title appear in the top left corner of the empty hole. *Uses allstates.dta*



```
scatter ownhome borninstate, by(nsw, title("% own home" "by %
born in state", ring(0) position(5) width(65) height(40)
justification(left) alignment(top)))
```

Now that we have aligned the text as we would like, we can remove the box by omitting the `box` option.
Uses allstates.dta

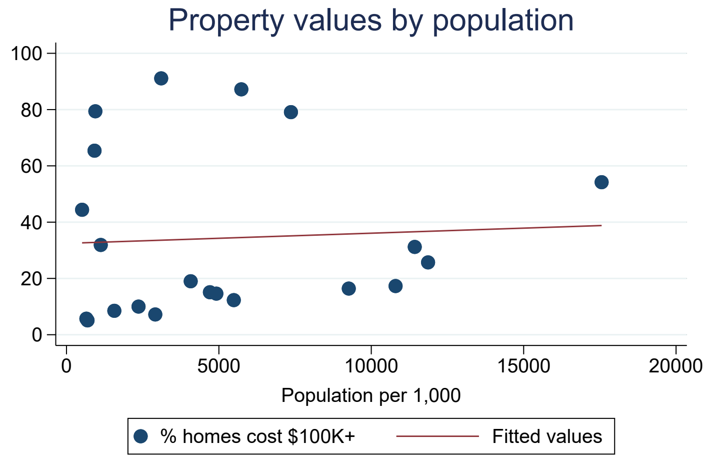


8.12 More options controlling the display of text

This section describes additional options to control the display of text, such as how to display symbols (such as Greek letters), how to create subscripts and superscripts, and how to display text in bold or italics. This section also illustrates how to select from among different fonts. I conclude this section by illustrating how you can specify Unicode characters within graph commands to display a wide variety of different characters. We begin by looking at examples illustrating the display of symbols.

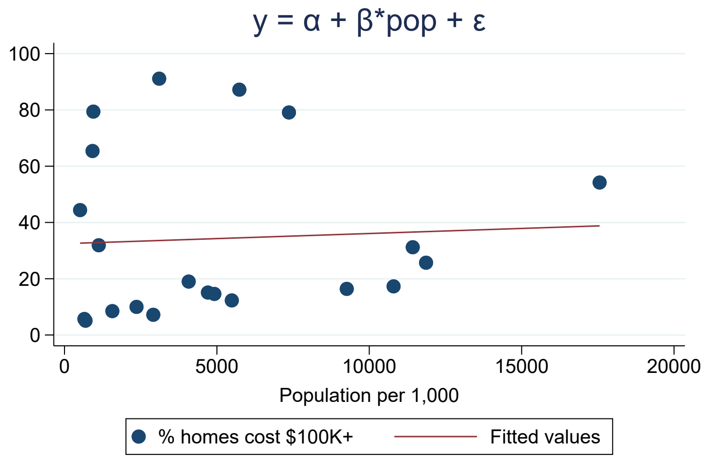
```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title(Property values by population)
```

Consider this graph that includes a scatterplot with a fit line from a linear regression. We might want to include the regression equation for the linear regression as a title. We can do so as shown in the next example. *Uses allstatesn.dta*



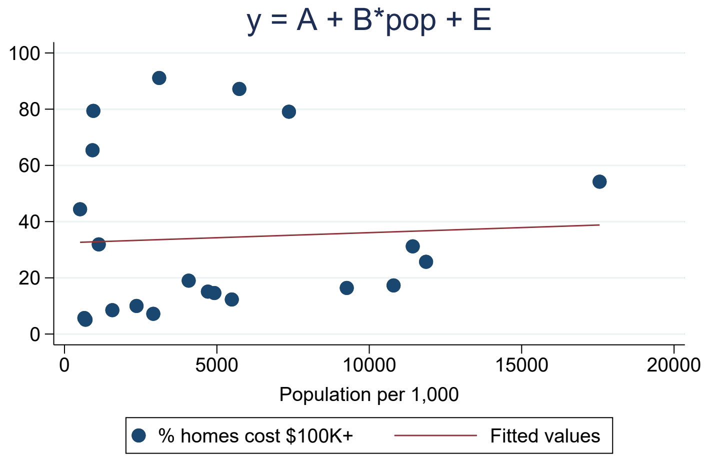
```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title(y = {&alpha} + {&beta}*pop + {&epsilon})
```

Note how the `{&alpha}` in the title is replaced with lowercase α (alpha). Likewise, `{&beta}` is replaced with lowercase β (beta), and `{&epsilon}` is replaced with lowercase ϵ (epsilon). Stata can display the full Greek alphabet (uppercase and lowercase), as well as a variety of math symbols. You can see `help text` for more details. *Uses allstatesn.dta*



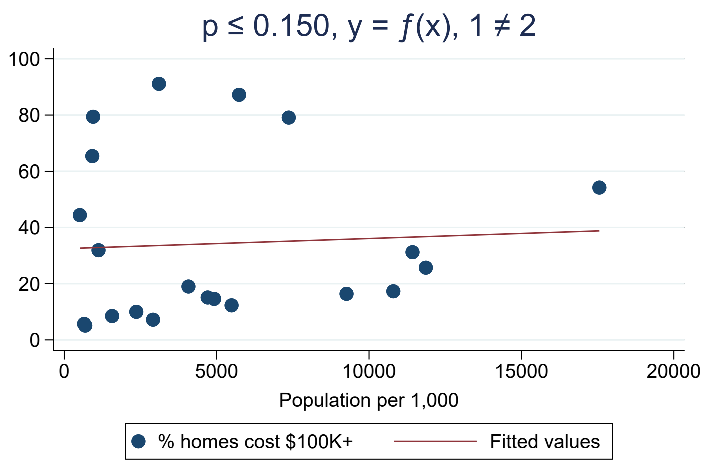
```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title(y = {&Alpha} + {&Beta}*pop + {&Epsilon})
```

Although not customary, we could replace the Greek letters with their uppercase counterparts by making the first letter uppercase, that is, replacing `{&alpha}` with `{&Alpha}`. *Uses allstatesn.dta*



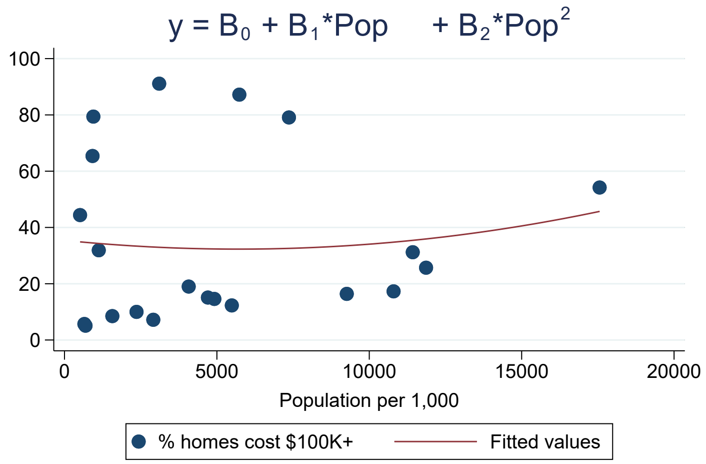
```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title("p {&le} 0.150, y = {&function}(x), 1 {&ne} 2")
```

In addition to the full Greek alphabet, Stata can display a wide variety of symbols. For example, I have inserted some nonsense in the title of the graph to illustrate three symbols: `{&le}` (less than or equal to), `{&function}` (a function of), and `{&ne}` (not equal to). You can see `help text` for a complete list of the symbols Stata can display. *Uses allstatesn.dta*



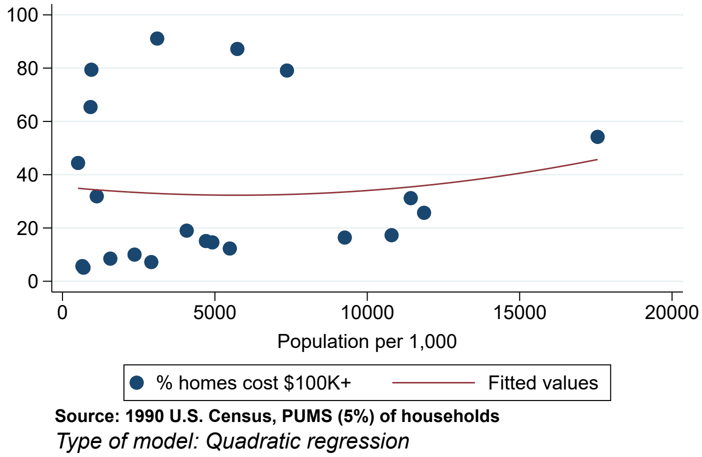
```
graph twoway (scatter propval100 popk) (qfit propval100 popk),
title(y = {&Beta}{sub:0} + {&Beta}{sub:1}*Pop + {&Beta}
{sub:2}*Pop{sup:2})
```

In this example, property values are modeled as a linear and quadratic function of population with the regression equation displayed in the title. Note how `{sub:0}` is used to display a 0 that is subscripted. Also note how `{sup:2}` displays squared (the number 2 superscripted). *Uses allstatesn.dta*



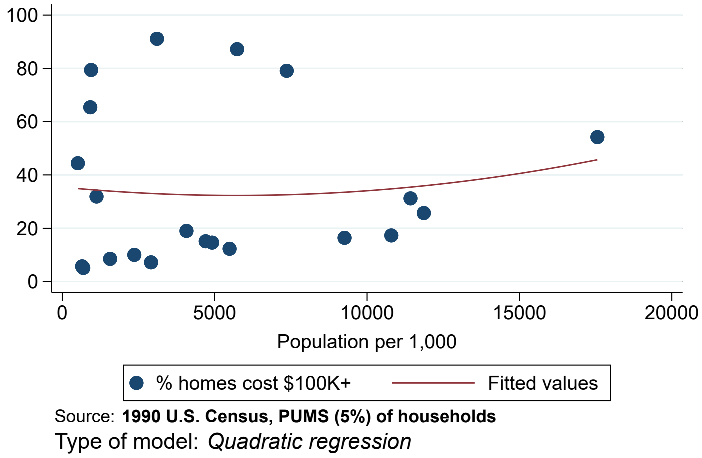
```
graph twoway (scatter propval100 popk) (qfit propval100 popk),
note("{bf:Source: 1990 U.S. Census, PUMS (5%) of households}")
caption({it:Type of model: Quadratic regression})
```

Suppose we want to display text as bold or as italics. In this example, the `note()` option specifies to display text in bold and the `caption()` option specifies to display text in italics. As you can see, the text that is enclosed between `{bf:` and `}` is displayed in bold. Likewise, the text that is enclosed between {it: and } is displayed in italics. *Uses allstatesn.dta*



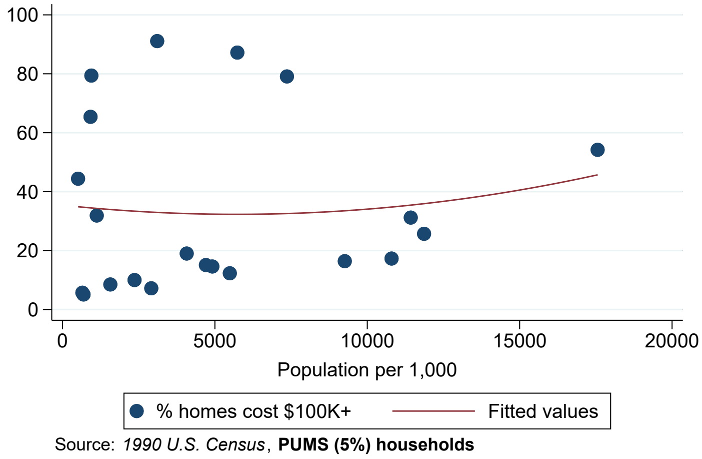
```
graph twoway (scatter propval100 popk) (qfit propval100 popk),
note("Source: {bf:1990 U.S. Census, PUMS (5%) of households}")
caption(Type of model: {it:Quadratic regression})
```

As shown in this example, you can also display just a portion of text using italics or bold. Part of the note is displayed in bold, and part of the caption is displayed in italics. *Uses allstatesn.dta*



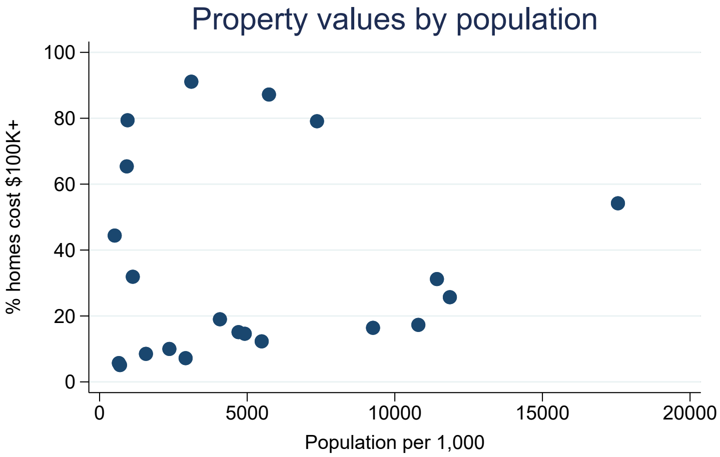
```
graph twoway (scatter propval100 popk) (qfit propval100 popk),
note("Source: {it:1990 U.S. Census}, {bf:PUMS (5%)
households}")
```

This example shows a note in which some of the text is displayed as normal text, some of the text is displayed in italics, and some of the text is displayed in bold. *Uses allstatesn.dta*



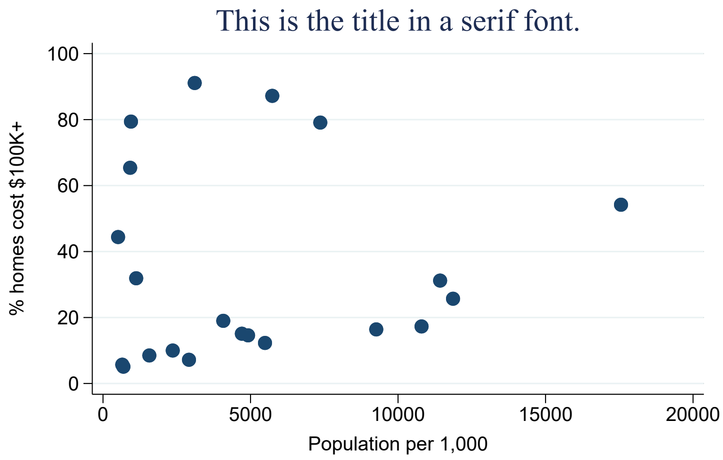
```
scatter propval100 popk, title(Property values by population)
```

Consider this basic graph that includes a title. In the following examples, we will see how we can specify different fonts. *Uses allstatesn.dta*



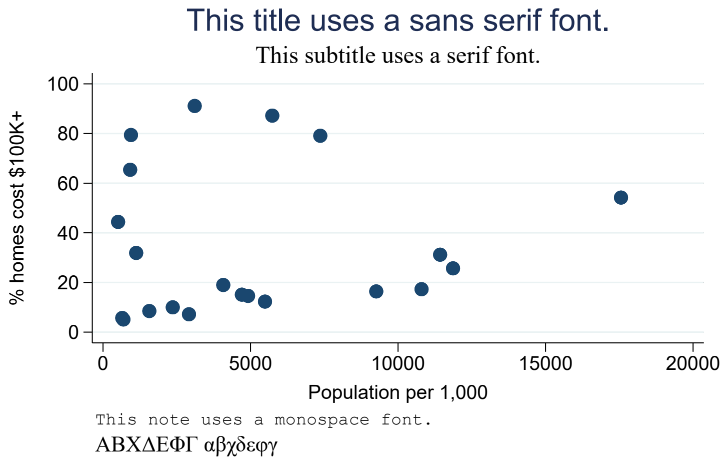
```
scatter propval100 popk, title({stSerif:This is the title in a
serif font.})
```

The title in this example is displayed using a serif font. The text that is enclosed between `{stSerif:` and `}` is displayed using a serif font. The font name, `stSerif`, is a built-in font that is provided by Stata. *Uses allstatesn.dta*



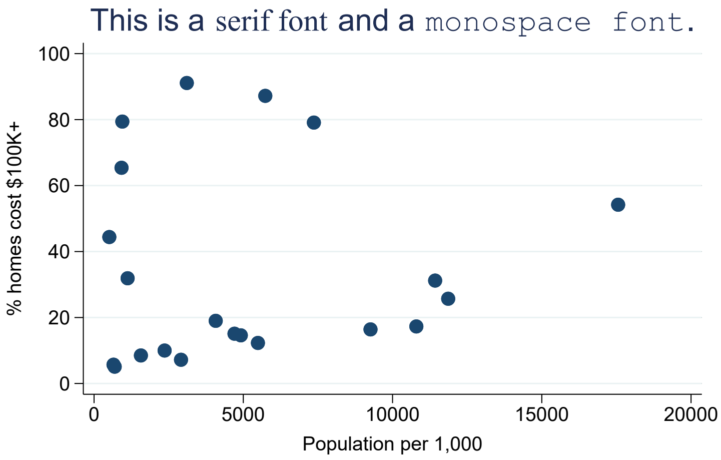
```
scatter propval100 popk, title({stSans:This title uses a sans
serif font.}) subtitle({stSerif:This subtitle uses a serif
font.}) note({stMono:This note uses a monospace font.})
caption({stSymbol:ABCDEFG abcdefg})
```

This example illustrates the four built-in fonts that are available within Stata. These fonts are named `stSans`, `stSerif`, `stMono`, and `stSymbol`. The title is displayed using the `stSans` font, the subtitle using `stSerif`, the note using `stMono`, and the caption using `stSymbol`. The default font for displaying text is `stSans`. *Uses allstatesn.dta*



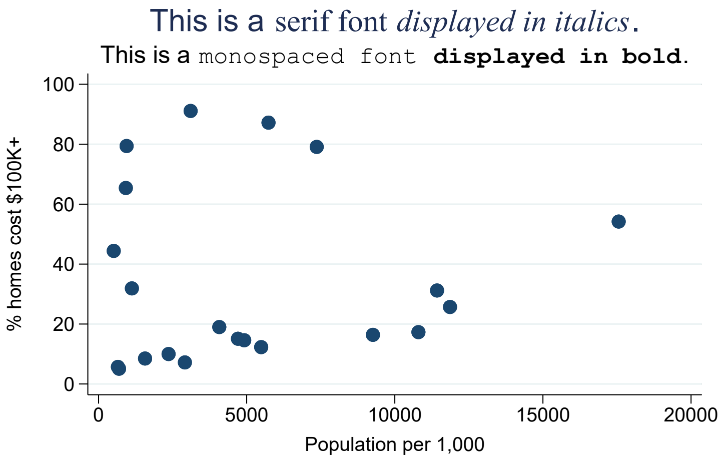
```
scatter propval100 popk, title(This is a {stSerif:serif font}
and a {stMono:monospace font}.)
```

This example shows that you can alternate fonts within the title. *Uses allstatesn.dta*



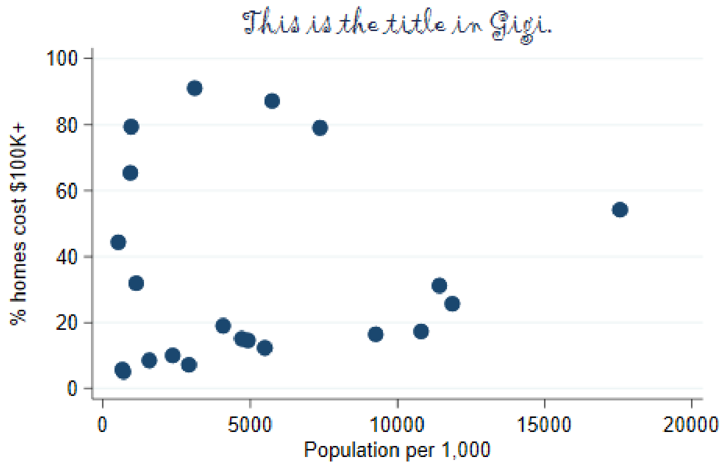
```
scatter propval100 popk, title(This is a {stSerif:serif font
{it:displayed in italics}}.) subtitle(This is a
{stMono:monospaced font {bf:displayed in bold}}.)
```

Note how this example both specifies a font type and displays some of the text in italics and some of the text in bold. This illustrates that you can specify a font type combined with italics or combined with bold. *Uses allstatesn.dta*



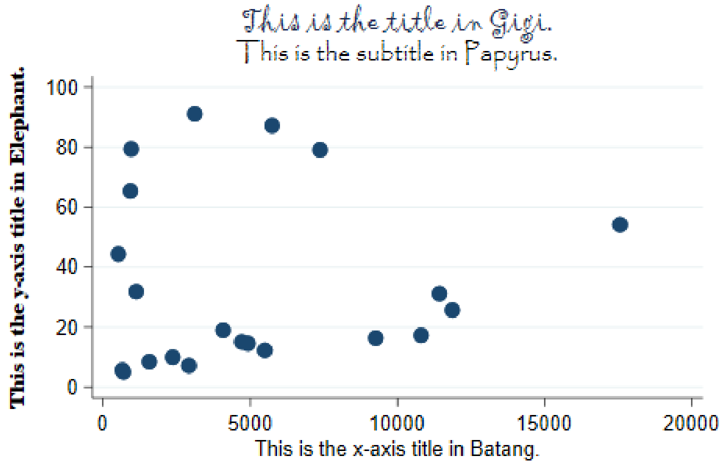
```
scatter propval100 popk, title({fontface Gigi:This is the
title in Gigi.})
```

In addition to the four built-in Stata fonts, you can select fonts that are available on your computer. For example, I was using my word processor on my Windows computer and saw that I have a font named `Gigi`. In this example, I specify that the title should be displayed using this font. *Uses allstatesn.dta*



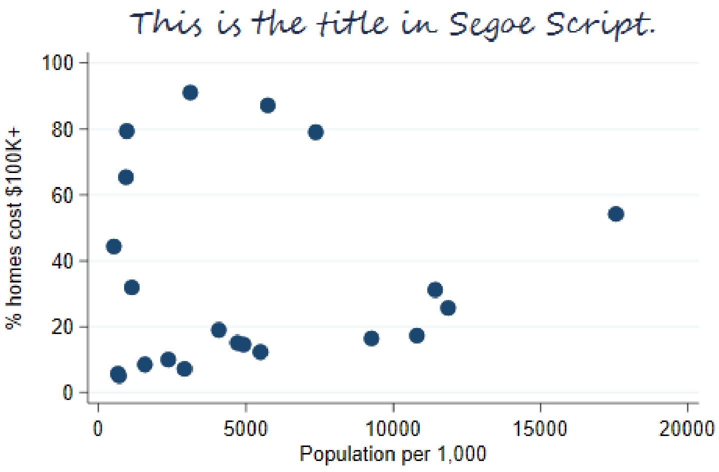
```
scatter propval100 popk, title({fontface Gigi:This is the
title in Gigi.}) subtitle({fontface Papyrus:This is the
subtitle in Papyrus.}) xtitle({fontface Batang:This is the x-
axis title in Batang.}) ytitle({fontface Elephant:This is the
y-axis title in Elephant.})
```

Further looking at the list of fonts available in my word processor, I found that I have fonts named `Gigi`, `Papyrus`, `Batang`, and `Elephant`. I am going to go crazy in this example and display the title in `Gigi`, the subtitle using the `Papyrus` font, the x-axis title using the `Batang` font, and the y-axis title using the `Elephant` font. *Uses allstatesn.dta*



```
scatter propval100 popk, title(`"{fontface "Segoe Script":This
is the title in Segoe Script.}"')
```

Sometimes, the name of a font contains a space within it. For example, I have a font on my computer named `Segoe Script`. To specify this font, I need to enclose the name within double quotes. Because this introduces a double quotation within the `title` option, the title is surrounded by compound quotes; that is, the title begins with `` `" `` and ends with `"'` (see `help quotes`). *Uses allstatesn.dta*



I would like to underscore that these examples use custom fonts that were available on my computer. The fonts that you have available on your computer will likely be different than mine. There are several ways to discover the fonts that you have on your computer, but perhaps the easiest is to use your word processor and look at the font options that it provides.

I will conclude this section by illustrating how you can insert Unicode characters into your graph commands to display a wide variety of symbols, Greek characters, or even emojis. Stata allows you to include Unicode characters in your graph just like any other ordinary characters like A to Z.

On my macOS computer, I can insert commonly used Unicode symbols into Stata very simply. With my cursor in either the Stata Command window or Do-file Editor, I can click on the **Edit** menu and then choose **Emoji & Symbols**. That brings up a window called *Character Viewer*. I can search among categories of characters, or I can search for characters in the *Search* window. Once I find a character I like, I can double-click on it, and it will be pasted into the active window within Stata. I created the `display` command below by inserting a grinning face, a frowning face, and a pineapple.

```
. display "Here is a grinning face  and a frowning face  and a pineapple "
Here is a grinning face  and a frowning face  and a pineapple
```

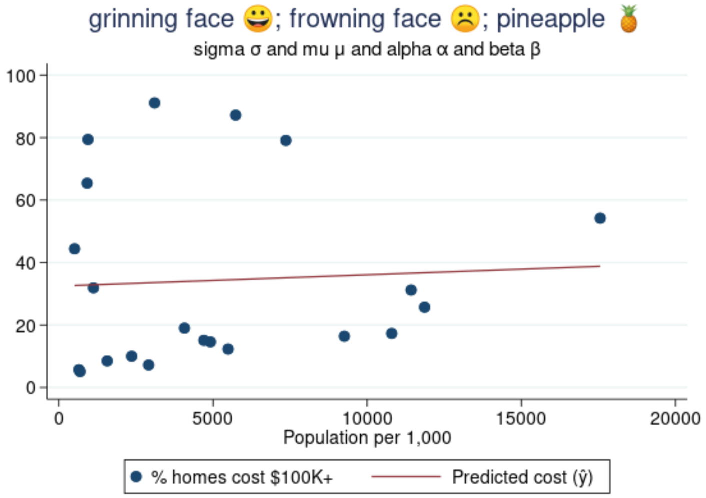
I next used the Search box within the *Character Viewer* to search for yhat, sigma, mu, alpha, and beta, yielding the `display` command below.

```
. display "Here is yhat ŷ and sigma  and mu  and alpha  and beta "
Here is yhat ŷ and sigma  and mu  and alpha  and beta
```

I can just as easily paste these characters into a `graph` command, as illustrated below.

```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title("grinning face ; frowning face ; pineapple ")
subtitle("sigma and mu and alpha and beta ") legend(label(2
"Predicted cost (ŷ)"))
```

In this graph, I pasted the grinning face, frowning face, and pineapple in the title. I pasted the sigma, mu, alpha, and beta in the subtitle, and I pasted the yhat in the `legend()` option. *Uses allstatesn.dta*



If you are using a non-macOS computer, I would recommend searching the Internet to see whether there is a feature that similarly eases the process of inputting Unicode characters. For example, https://en.wikipedia.org/wiki/Unicode_input describes methods of inputting Unicode using different computer systems.

If you cannot find a simpler method for inputting Unicode characters, you can use this method for manually searching for and entering Unicode characters into your graph commands. Let me walk you through the steps for inserting a grinning face into the title of a graph. My first step is to search the Internet for the Unicode value for a grinning face. I searched for “Unicode UTF-32 grinning face”, and I found a webpage that described a symbol called *grinning face with smiling eyes*. The page shows numerous codes for specifying that symbol. I want the UTF-32 (hexadecimal) value, which I found as `0x0001F601`. I use the `display` command below to display that symbol.

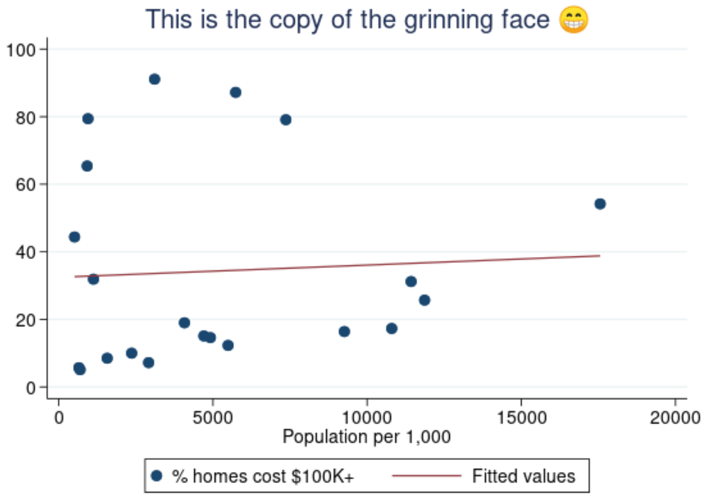
```
. display ustrunescape("\U0001F601")
```

Note that I removed the `0x` prefix and instead started with `\U` (note the uppercase U). I counted to make sure I had a total of eight digits. Stata is expecting exactly eight digits after `\U`.

I can copy the grinning face from the Results window into the Do-file Editor (or Command window). The `graph` command below was created by pasting the grinning face from the Results window into the `title()` option.

```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title("This is the copy of the grinning face ")
```

I copied the grinning face from the Results window and pasted it into the title of my graph. Now my title ends with a grinning face. *Uses allstatesn.dta*

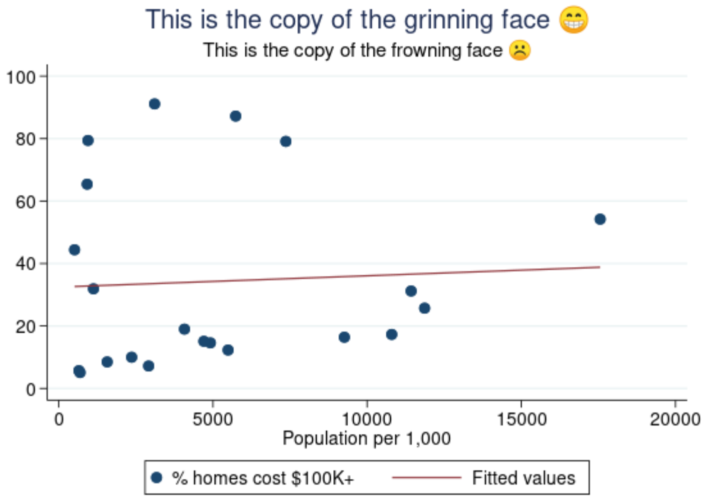


Let's now search for a frowning face. I searched for "Unicode UTF-32 frowning face" and found a symbol described as *slightly frowning face*. The webpage listed its UTF-32 (hex) code as `0x0001F641`. I use the `display` command to show that symbol.

```
. display ustrunescape("\U0001F641")
```

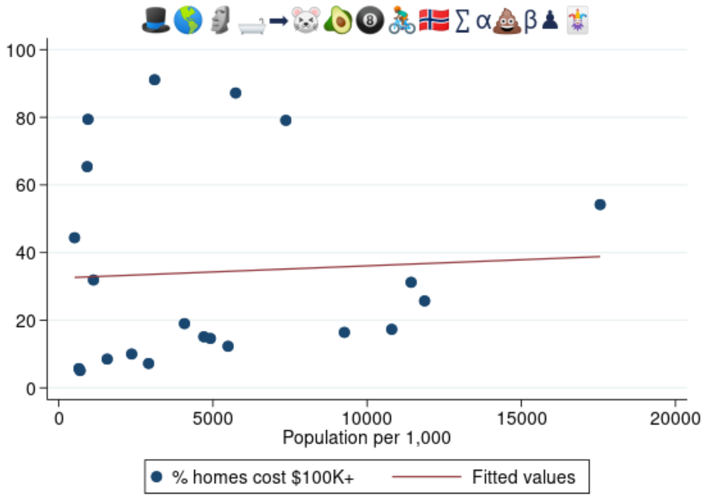
```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title("This is the copy of the grinning face ") subtitle("This
is the copy of the frowning face ")
```

This graph uses the grinning face in my title and the frowning face in the subtitle. *Uses allstatesn.dta*



```
graph twoway (scatter propval100 popk) (lfit propval100 popk),
title("")
```

The title of this graph whimsically illustrates more characters that you can specify via Unicode. *Uses allstatesn.dta*



You can specify an incredibly wide array of characters via Unicode. Wikipedia has an article about the list of all Unicode characters; see `https://en.wikipedia.org/wiki/List_of_Unicode_characters`. You can learn more about Unicode by visiting the webpage of the Unicode Consortium at `https://home.unicode.org` and seeing the Wikipedia page on Unicode at `https://en.wikipedia.org/wiki/Unicode`.

Chapter 9
Standard options available for all graphs

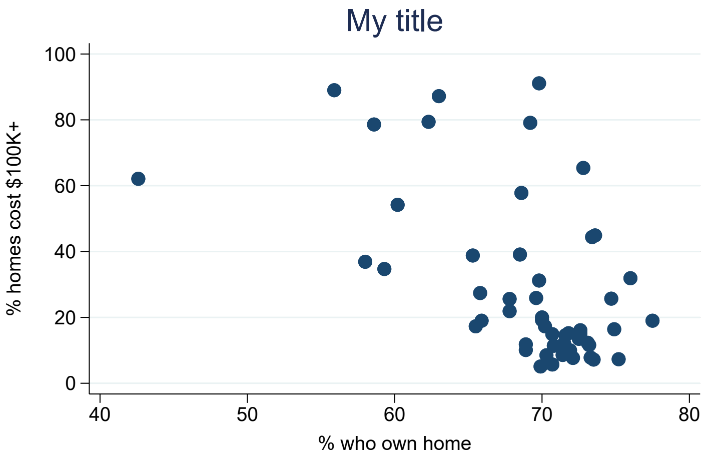
This chapter discusses a class of options Stata refers to as *standard options*, because these options can be used in all graphs. This chapter begins by discussing options that allow us to add or change the titles in the graph and then shows us how to use schemes to control the overall look and style of our graph. The next section demonstrates options for controlling the size of the graph and the scale of items within graphs. The chapter concludes by showing options that allow us to control the colors of the plot region, the graph region, and the borders that surround these regions. For more details, see `help std_options`.

9.1 Creating and controlling titles

Titles are useful for providing more information that explains the contents of a graph. Stata includes four standard options for adding explanatory text to graphs: `title()`, `subtitle()`, `note()`, and `caption()`. This section shows how to use these options to add titles and how to customize the title's content and placement. For more information about customizing the appearance of such titles (for example, color, size, orientation), see Options : Textboxes (section 8.11). For more information about titles, see `help title_options`.

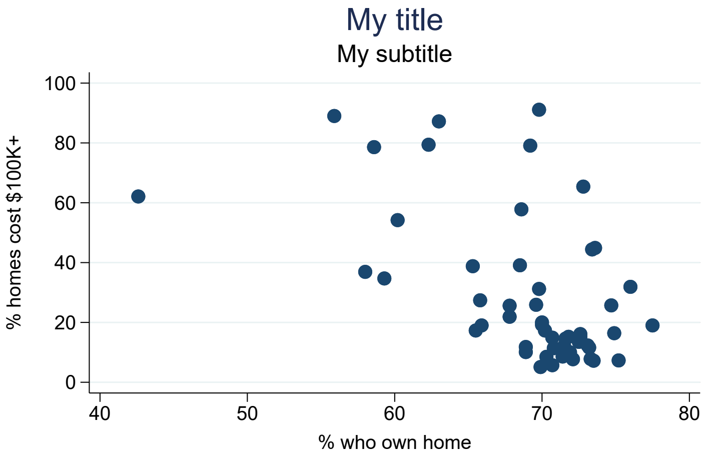
```
scatter propval100 ownhome, title("My title")
```

The `title()` option adds a title to a graph. Here we add a simple title to the graph. Although the title includes quotes, we could have omitted them. Later, we will see examples where the quotes become important. *Uses allstates.dta*



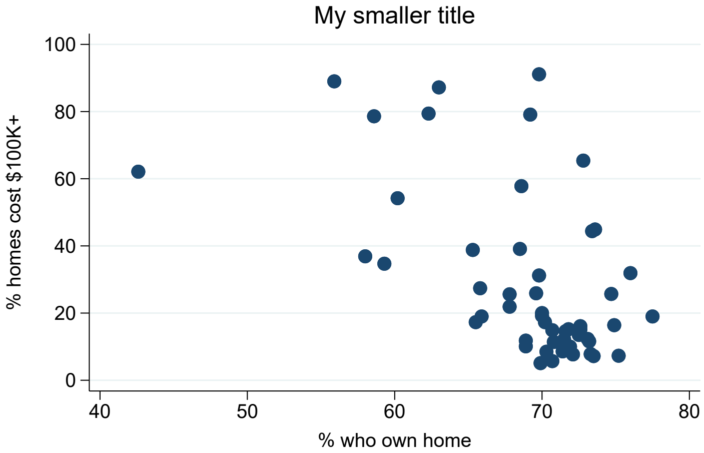
```
scatter propval100 ownhome, title("My title") subtitle("My
subtitle")
```

The `subtitle()` option adds a subtitle to a graph. The subtitle, by default, appears below the title in a smaller font. *Uses allstates.dta*



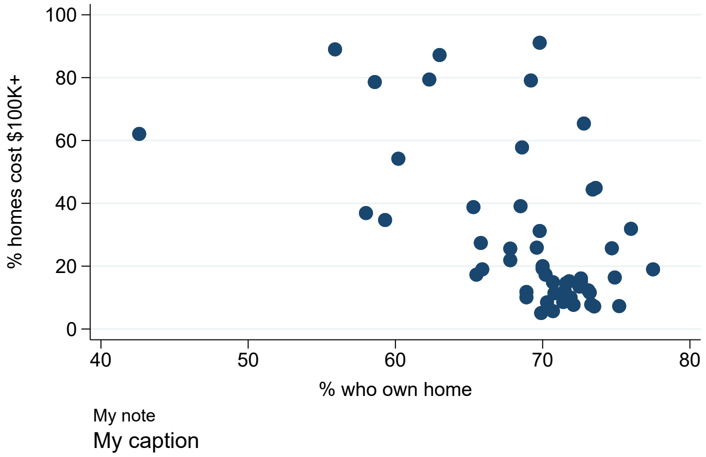
```
scatter propval100 ownhome, subtitle("My smaller title")
```

We do not have to specify `title()` to specify `subtitle()`. For example, we might want a title that is smaller than a regular title, so we could specify `subtitle()` alone. *Uses allstates.dta*



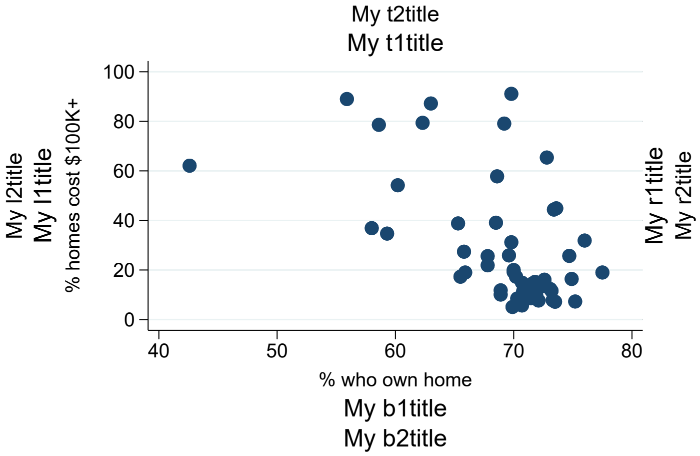
```
scatter propval100 ownhome, caption("My caption") note("My
note")
```

Here the `caption()` option adds a small-sized caption in the lower left corner, and the `note()` option places a smaller-sized note in the bottom left corner. If we specify both options, the note appears above the caption. Both options do not need to be included in the same graph. *Uses allstates.dta*



```
scatter propval100 ownhome, t1title("My t1title") t2title("My
t2title") b1title("My b1title") b2title("My b2title")
l1title("My l1title") l2title("My l2title") r1title("My
r1title") r2title("My r2title")
```

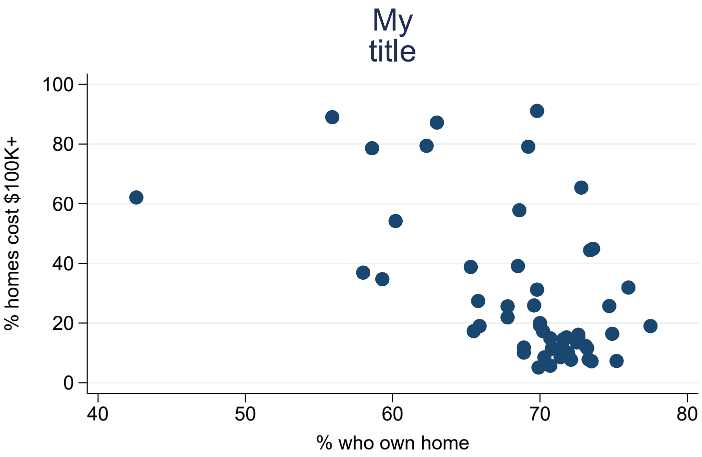
Although these are not as commonly used, Stata offers several other title options for titling the top of the graph (`t1title()` and `t2title()`), the bottom of the graph (`b1title()` and `b2title()`), the left side of the graph (`l1title()` and `l2title()`), and the right side of the graph (`r1title()` and `r2title()`). *Uses allstates.dta*



Stata gives you considerable flexibility in the placement of these titles, notes, and captions, as well as control over the size, color, and orientation of the text. This is illustrated below by using the `title()` option, but the same options apply equally to the `subtitle()`, `note()`, and `caption()` options.

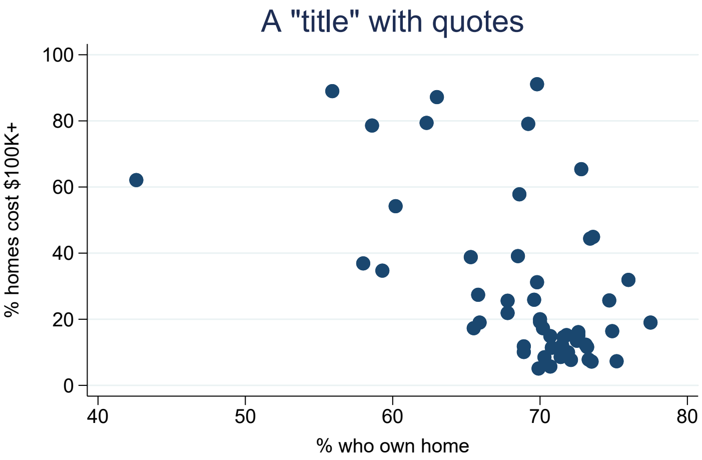
```
scatter propval100 ownhome, title("My" "title")
```

In this example, we use multiple sets of quotes in the `title()` option to tell Stata that we want the title to appear on two separate lines. *Uses allstates.dta*



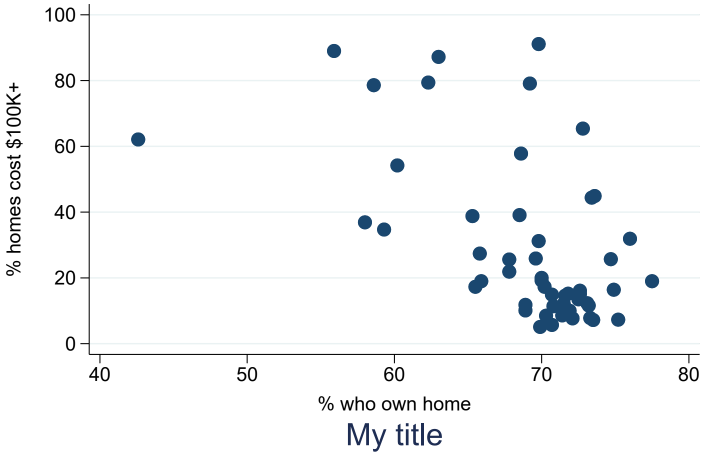
```
scatter propval100 ownhome, title(`"A "title" with quotes"')
```

This example illustrates that we can have quotation marks in the `title()` option, as long as we open the title with `` `" `` and close it with `"'`. (The open single quote is often located below the tilde on your keyboard, and the close single quote is often located below the double quote on your keyboard.) *Uses allstates.dta*



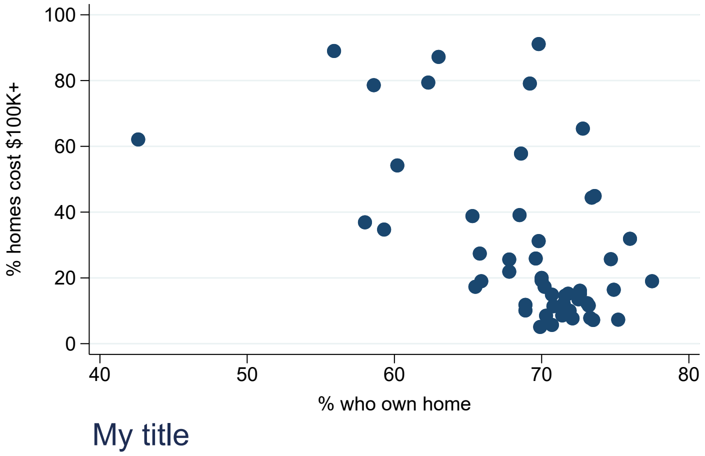
```
scatter propval100 ownhome, title("My title", position(6))
```

The `position()` option changes the position of the title. Here we place the title in the bottom of the graph by indicating that it should be at the six o'clock position. See Styles : Clockpos (section 10.3) for more details. *Uses allstates.dta*



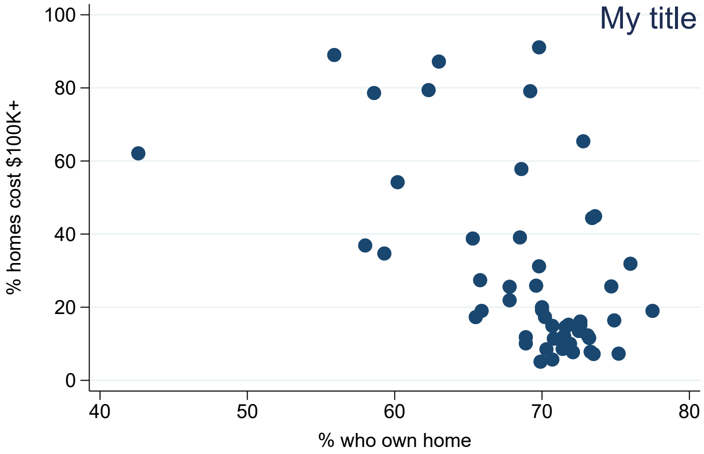
```
scatter propval100 ownhome, title("My title", position(7))
```

In this example, we place the title in the bottom left corner of the graph by indicating that it should be at the seven o'clock position. *Uses allstates.dta*



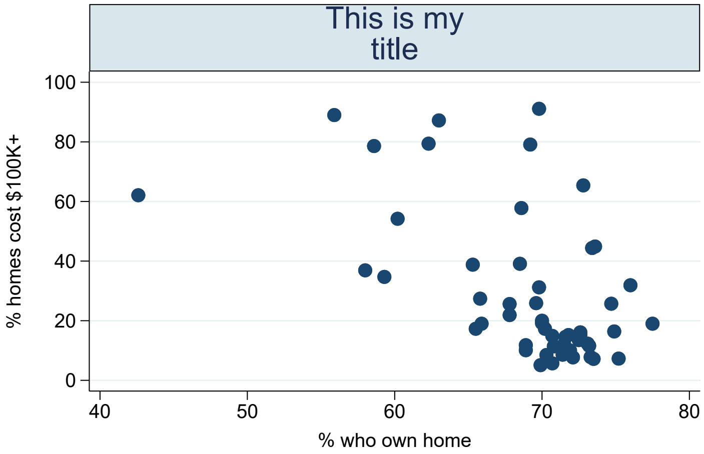
```
scatter propval100 ownhome, title("My title", position(1)
ring(0))
```

Here we not only use the `position()` option to place the title at the one o'clock position, but we also use the `ring(0)` option to place the title inside the plot region. Higher values for `ring()` place the item farther away from the plot region. Imagine concentric rings around the plot area with higher values corresponding to the rings that are farther from the center. *Uses allstates.dta*



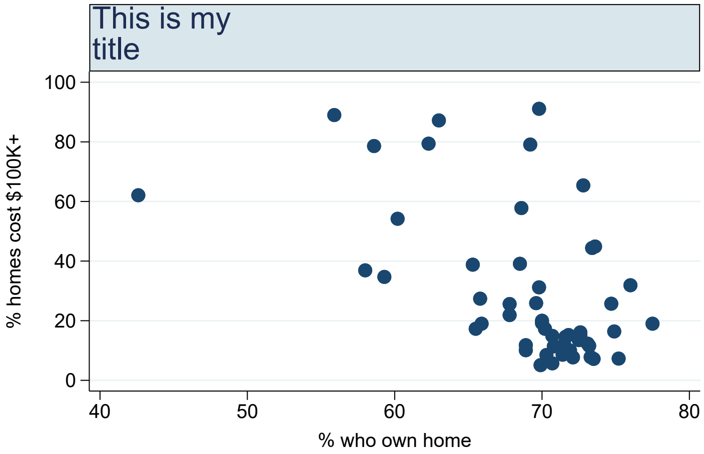
```
scatter propval100 ownhome, title("This is my" "title", box
bexpand)
```

Because titles, subtitles, notes, and captions are considered textboxes, we can use the options associated with textboxes to customize their display. Here we add a box and expand it to fill the width of the plot region. *Uses allstates.dta*



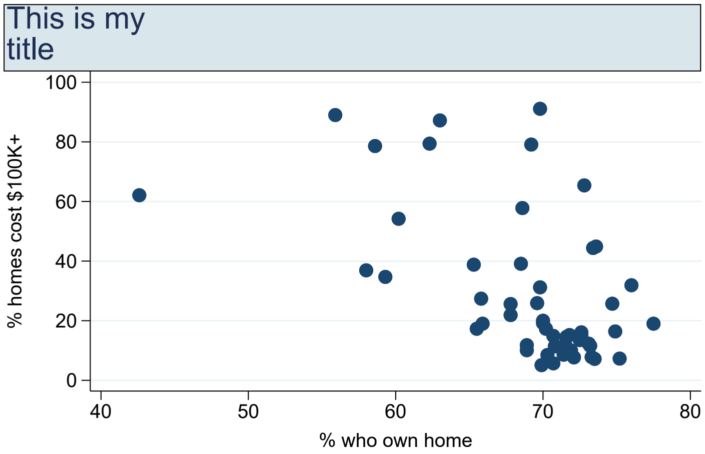
```
scatter propval100 ownhome, title("This is my" "title", box
bexpand justification(left))
```

We can use the `justification(left)` option to left justify the text inside the box. Note the difference between the `position()` option (illustrated in previous examples), which positions the textbox, and the `justification()` option, which justifies the text within the textbox. *Uses allstates.dta*



```
scatter propval100 ownhome, title("This is my" "title", box
bexpand justification(left) span)
```

We can use the `span` option to make the box span the entire width of the graph. There are numerous other textbox options that we can use with titles; see Options : Textboxes (section 8.11) and `help textbox_options` for more details. *Uses allstates.dta*



9.2 Using schemes to control the look of graphs

Schemes control the overall look of Stata graphs by providing default values for many graph options. Schemes can drastically alter the look of your graph. Selection of a graph scheme is the most impactful decision you will make on determining the look, style, and philosophy of your graph. In all other parts of this book, I focused on showing one scheme because I wanted the examples in the graphs to reflect the impact of the specified options and only the specified options. Now, I want to show you a variety of schemes that you can choose from. These schemes come from three different sources: official schemes that ship with Stata (see section 9.2.1); schemes that were written by members of the Stata community (see section 9.2.2); and schemes that accompany this book (see section 9.2.3).

9.2.1 Schemes included with Stata

Several schemes included with Stata. In the following examples, I am going to illustrate what I believe to be the five most common and distinctive schemes that ship with Stata: `s2color`, `s1color`, `s2mono`, `s1mono`, and `economist`. There are additional official Stata graph schemes that I will not illustrate because they are very similar to one of the prior five schemes. Namely, I will not illustrate the `s2gcolor`, `s2manual`, `s2gmanual`, `s2color8`, `s1manual`, or `s1rcolor` scheme. See `help schemes` for more details both about the schemes I included and the schemes I excluded.

9.2.2 Community-contributed schemes

In addition to the schemes installed with Stata, I would like to introduce you to several schemes created by the Stata community. These are described below, with instructions on downloading the latest version of the schemes to your computer. The location of the latest version might change after the publication of this book. You can use the `search` command within Stata to search for the latest versions of these schemes. Also, you can check the website for this book for updated information on downloading the latest version of these schemes [see Introduction : Online supplements (section 1.1)].

Graph schemes sensitive to color vision deficiency

Daniel Bischof has created a family of four schemes to create graphs that include versions that are sensitive to people with color vision deficiency. He provides the `plotplain` and `plottig` schemes, each of which have its own specific look. Then, he provides versions of each of these schemes that are easier to read for people with color vision deficiency, namely, `plotplainblind` and `plottigblind`. You can learn more about these schemes by seeing his article titled "New graphic schemes for Stata: plotplain and plottig", published in the *Stata Journal*, volume 17 number 3. Although these schemes are available with his article, I found the latest versions of these graphs could be obtained using the `ssc install` command shown below.

```
. ssc install blindschemes
```

For more details about these schemes, you can type `help plotplain`, `help plotplainblind`, `help plottig`, or `help plottigblind`.

The 538 family of schemes

Daniel Bischof has also created a family of three schemes that reflect the design of graphs shown on the very popular website https://fivethirtyeight.com. The schemes are named `538`, `538bw`, and `538w`. I found that I could download these schemes with the `ssc install` command shown below.

```
. ssc install g538schemes
```

You can then type `help 538` to learn more about these schemes.

The lean family of schemes

Svend Juul has created a family of lean graphs named `lean1` and `lean2`. They are documented in the article titled "Lean mainstream schemes for Stata 8 graphics", published in the *Stata Journal*, volume 3 number 3. I found that the latest update to these schemes was published in the *Stata Journal*, volume 4 number 3, which I could download using the following commands.

```
 net sj 4-3 gr0002_3
net install gr0002_3
```

You can then get more information about these schemes by typing `help scheme_lean1` and `help scheme_lean2`.

9.2.3 Schemes included with this book

I created four additional color schemes for this book and four corresponding black-and-white versions. The color versions are

- `vg_palec`—Graphs where bars/boxes/and piles use pale fill colors
- `vg_hollowc`—Graphs where markers/bars/boxes/pies use invisible fill areas
- `vg_lgndc`—Graphs with the legend on the left in one column
- `vg_samec`—Graphs with the same fill color

There are four monochrome (black-and-white) versions of these schemes, named `vg_palem`, `vg_outm`, `vg_lgndm`, and `vg_samem`. See Introduction : Online supplements (section 1.1) for information about downloading these schemes.

As I mentioned in another part of the book, these schemes are like a single-purpose kitchen tool. I describe the schemes and show examples of their single-purpose use in section 1.4.3 . These schemes make occasional appearances in the book, showing off their single-purpose benefit. You can use the subject index to find such instances of their use—for example, the index entry for **schemes, vg_lgndc** would direct you to all the examples where this scheme was used throughout the book.

I included these schemes for one other purpose—to provide you with examples of simple schemes that do simple things and that would be a useful springboard for learning how to create a custom scheme of your own (see section 9.2.6).

I believe that the best way to learn about schemes is to see examples of a single graph shown using different schemes. This allows you to compare the graphs and see how the presentation of the graph changed when using a different scheme. To that end, I have created three examples that I will use for illustrating the schemes above. These examples are the following:

- Example #1: An overlaid scatterplot with fit lines
- Example #2: An overlaid scatterplot with fit lines and confidence region
- Example #3: A bar chart

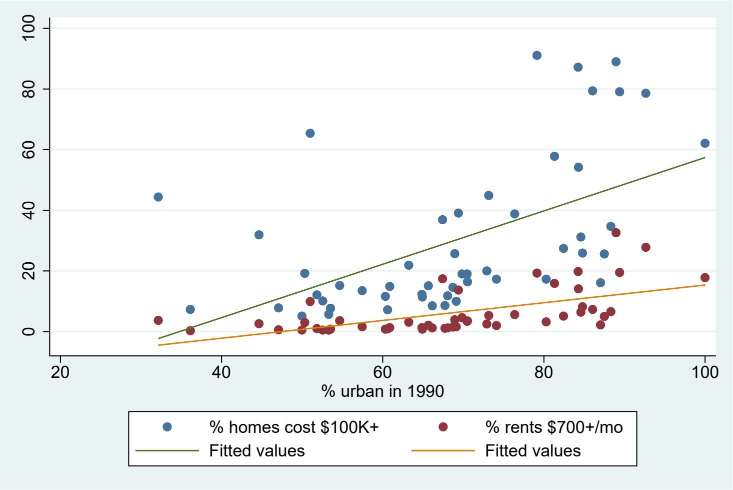
Each of the Stata schemes and community-contributed schemes will be illustrated using these examples. (To repeat—the schemes designed for this book are illustrated in section 1.4.3 and will not be illustrated below.)

9.2.4 Example #1: An overlaid scatterplot with fit lines

This example illustrates two overlapping scatterplots, each showing a linear fit line. This graph will illustrate the look of markers, linear fit lines, and how a legend is displayed. This example will specifically illustrate how schemes differ in the way the markers are displayed, how fit lines are displayed, and how a legend is displayed. Additionally, this example will convey the overall look of different schemes. I will often comment on whether shading is present in the *plot region* or the *graph region*. In the scatterplot below, the *plot region* is the white part of the graph where the markers and fit lines are plotted. The *graph region* is the light blue area where the axis labels, axis titles, and graph titles are displayed. In the example below, the plot region is white and includes horizontal grid lines. The graph region is shown in light blue, and the legend is displayed with a white background.

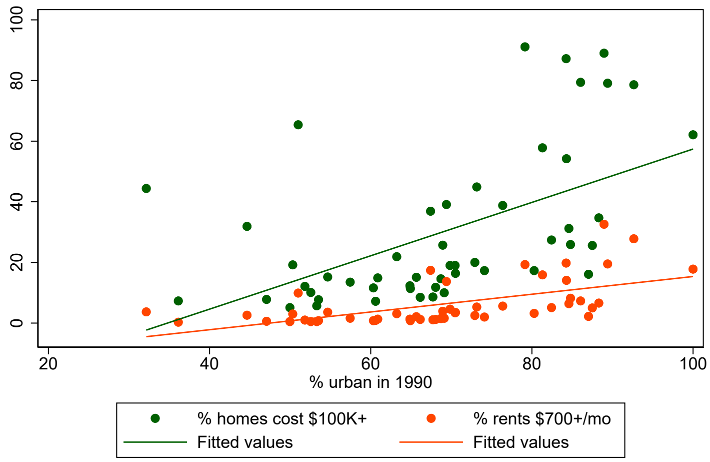
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(s2color)
```

This is an example of the `twoway` command, which I will use throughout this section. It plots home costs (in blue) and rents (in red) with a fit line for each (in blue and red)—these colors match because of the `pcycle(2)` option (see the example in section 2.11 . The plot region is white, the graph region is light blue, and the y-axis labels are at a 90-degree angle. *Uses allstates.dta*



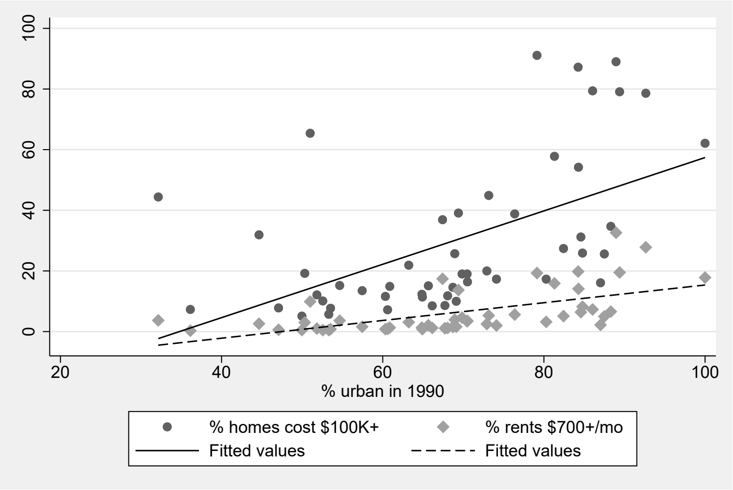
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(s1color)
```

The `s1color` scheme uses different colors for the markers and fit lines. More notably, the plot region and graph region are completely white—with no background color or any grid lines. Otherwise, the `s1color` and `s2color` schemes are rather similar. *Uses allstates.dta*



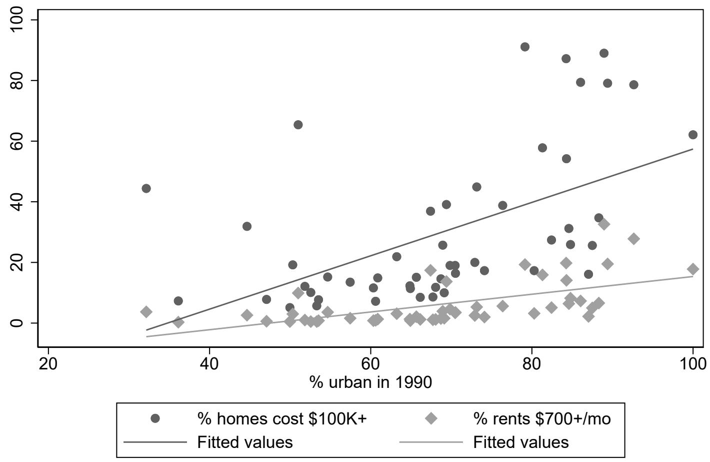
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(s2mono)
```

The `s2mono` scheme is a black-and-white version of the `s2color` scheme. Note how the symbols differ in gray scale and size and the fit lines differ in their patterns. *Uses allstates.dta*



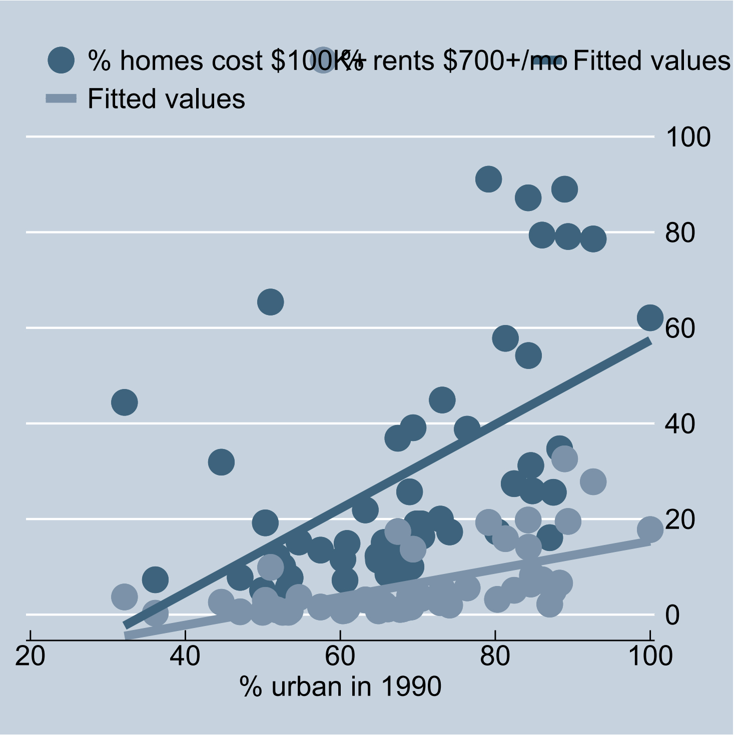
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(s1mono)
```

The `s1mono` scheme is the black-and-white counterpart to the `s1color` scheme. The look of those two schemes are very similar, except for the display of the markers and fit lines. The markers are displayed using different shapes and shades of gray, and the fit lines are shown using differing shades of gray. *Uses allstates.dta*

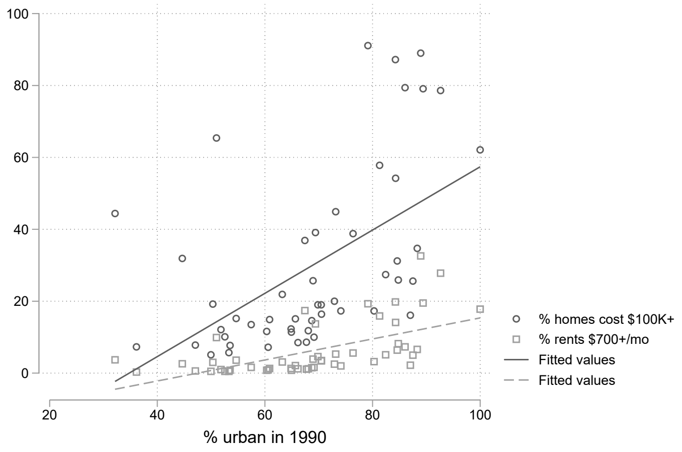


```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(economist)
```

The `economist` scheme has a completely different look than the `s1` and `s2` family of schemes. The entire graph is shown with a grayish background color. The markers and fit lines are far larger and shown using shades of blue. The legend for the graph is shown at the top, and the y axis is positioned at the right using unrotated axis labels. *Uses allstates.dta*

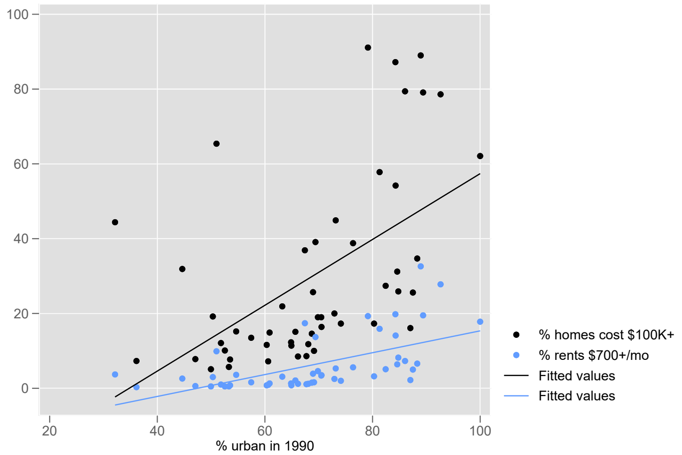


```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(plotplain)
```



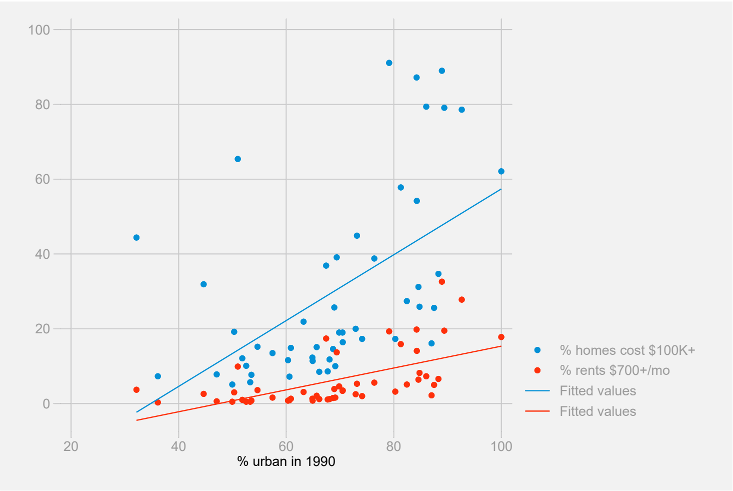
The `plotplain` scheme lives up to its name, creating a plain graph (that is meant as a compliment!). The home costs are graphed using small hollow circles, and the rents are graphed using small hollow squares. There is no shading, and the gridlines are faint. The legend is shown to the right of the graph, and the y-axis labels are not rotated. The `plotplainblind` scheme produces a similar looking graph, so it is omitted to save space. See section 9.2.2to download this scheme. *Uses allstates.dta*

```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(plottig)
```



The `plottig` scheme displays the plot region with a gray background and white grid lines and displays the graph region in white. The markers for home costs and rents are shown using small black solid circles and light-blue solid circles, respectively. The fit lines match the marker colors. The legend is shown to the right of the plot region, and the y-axis labels are not rotated. The graph using `plotplainblind` is similar, so it omitted to save space. See section 9.2.2to download this scheme. *Uses allstates.dta*

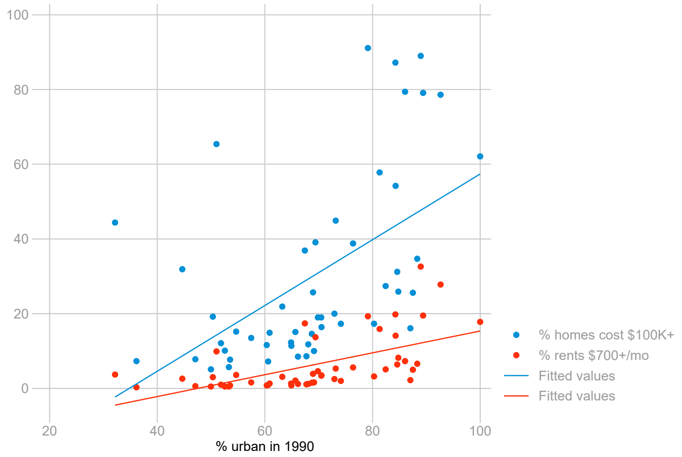
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(538)
```



The `538` scheme uses a gray background for the entire graph (both the plot region and graph region), and grid lines are displayed in dark gray. The markers and fit line for housing costs are shown in blue, and the markers and fit line for rents are shown in red. The legend is shown to the right of the graph, and the y-axis labels are not rotated. See section 9.2.2to download this scheme. *Uses allstates.dta*

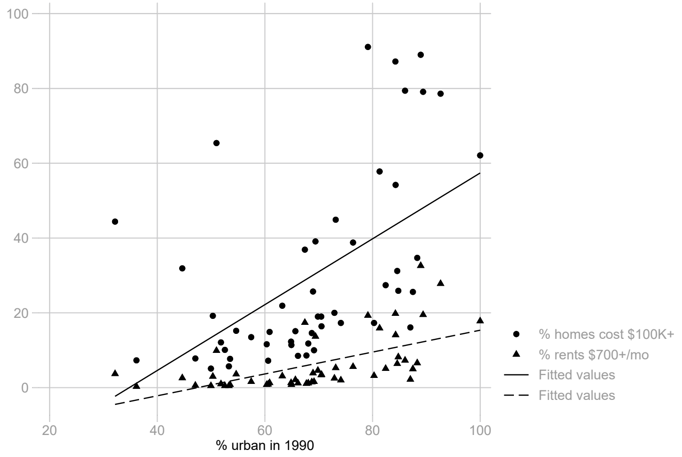
```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(538w)
```

Comparing the graph produced by the `538w` scheme with the previous graph, we can see that these two graphs are nearly identical. The only difference is that the background is displayed in white (compared with gray in the prior example). See section 9.2.2to download this scheme. *Uses allstates.dta*

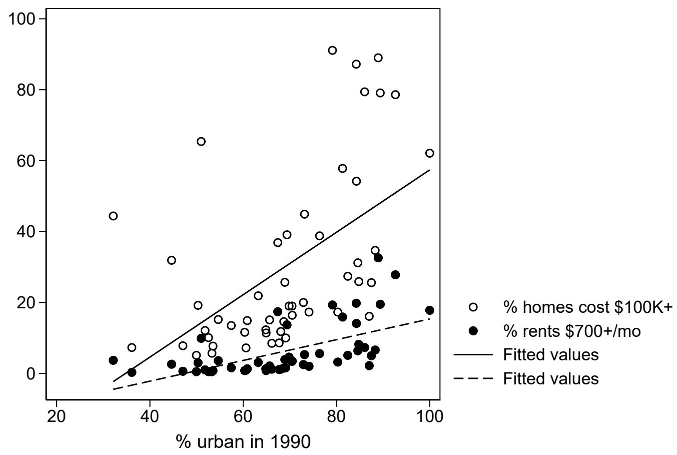


```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(538bw)
```

The `538bw` scheme is the black-and-white equivalent of the `538w` scheme. Home costs are shown with small solid circles and a solid fit line, while rents are shown with small solid triangles and a long dashed fit line. Otherwise, the look of this graph is much like the prior graph. See section 9.2.2to download this scheme. *Uses allstates.dta*

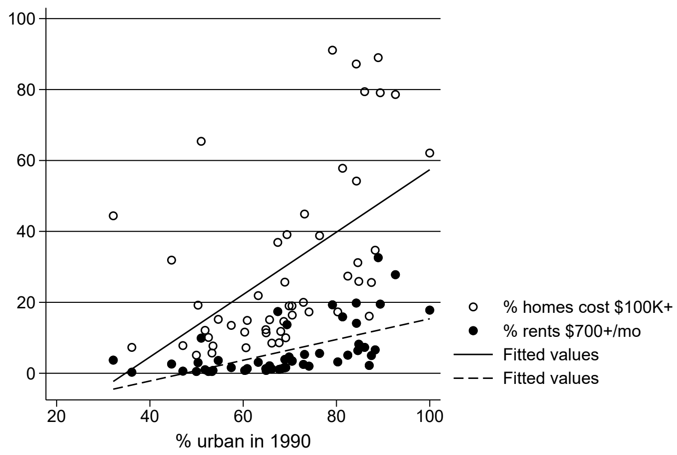


```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(lean1)
```



The background color for the `lean1` scheme is white and omits any grid lines, but the plot area is framed. Home costs are shown with hollow circles and a solid fit line, while rents are shown with solid black circles and a long dashed fit line. The legend is shown to the right of the graph, and the y-axis labels are not rotated. See section 9.2.2to download this scheme. *Uses allstates.dta*

```
twoway (scatter propval100 urban) (scatter rent700 urban)
(lfit propval100 urban) (lfit rent700 urban), pcycle(2)
scheme(lean2)
```



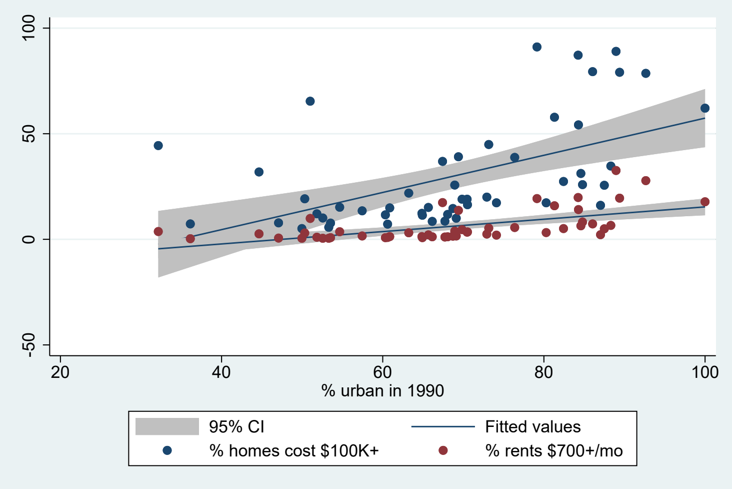
The markers and fit lines for home costs and rents are the same in this graph as the prior graph. The notable difference is that `lean2` includes horizontal grid lines and does not frame the plot region. See section 9.2.2to download this scheme. *Uses allstates.dta*

This section does not include examples of the schemes included with this book. Instead, see section 1.4.3, where those schemes are described and illustrated.

9.2.5 Example #2: An overlaid scatterplot with fit lines and confidence region

This example shows the same overlaid scatterplot as example 1 in section 9.2.4 with linear fit but now adds confidence regions using the `lfitci` command. I will draw attention to how the confidence region is shaded and what additional customizations are needed to ensure that the confidence regions do not conceal one another. I will also pay greater attention to the potential overlapping of markers. Note that some of these commands will use abbreviations because of space constraints.

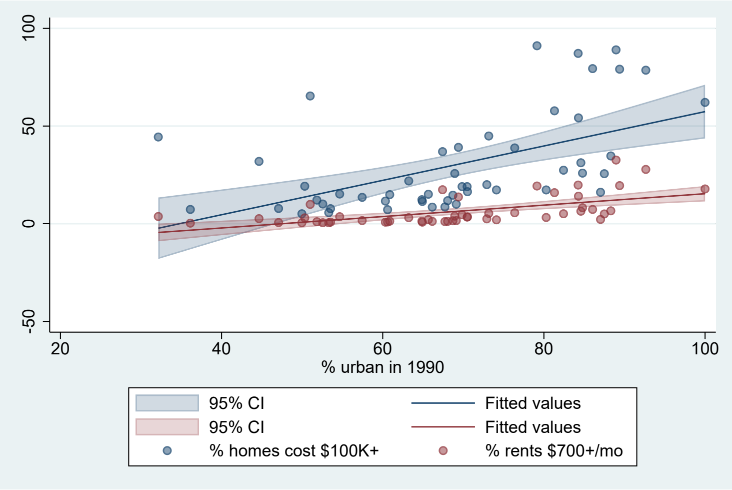
```
tw (lfitci propval100 urban) (lfitci rent700 urban) (scatter
propval100 urban) (scatter rent700 urban), pcycle(2)
scheme(s2color)
```



This example illustrates an overlaid scatterplot with linear fit and confidence regions using the `s2color` scheme. Notice how the confidence regions do not match the color of the markers and how the overlap is hard to distinguish. Similarly, if any red rent markers overlapped with any blue home cost marker, the red marker would conceal the blue marker. *Uses allstates.dta*

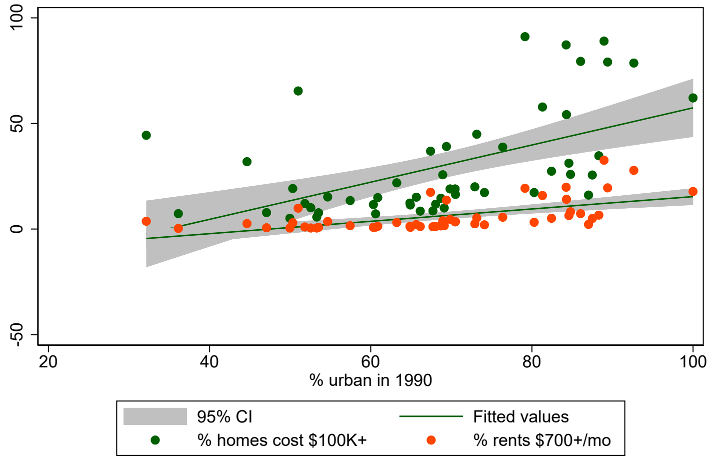
```
tw (lfitci propval100 urban, bcolor(navy%20) lcolor(navy))
(lfitci rent700 urban, bcolor(maroon%20) lcolor(maroon)) (scat
propval100 urban, mcolor(%50)) (scat rent700 urban,
mcolor(%50)), pcycle(2) scheme(s2color)
```

This example adds options to improve the appearance of the graph. Namely, each `lfitci` command uses the `lcolor()` option to make the color of the fit line match the markers and the `bcolor()` option to display each confidence region using the color to match the markers but with 20% opacity. Also, each `scatter` command uses the `mcolor(%50)` option to display the markers with 50% opacity. *Uses allstates.dta*



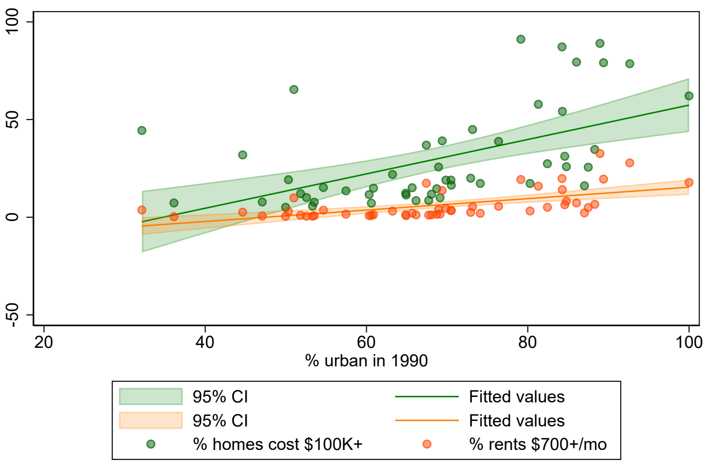
```
tw (lfitci propval100 urban) (lfitci rent700 urban) (scatter
propval100 urban) (scatter rent700 urban), pcycle(2)
scheme(s1color)
```

This example illustrates an overlaid scatterplot with linear fit and confidence regions using the `s1color` scheme. As with the `s2color` scheme, the confidence regions do not match the color of their markers, and the overlapping confidence regions are hard to distinguish. Similarly, if any orange rent markers overlapped with any green home cost marker, the orange marker would conceal the green marker. *Uses allstates.dta*



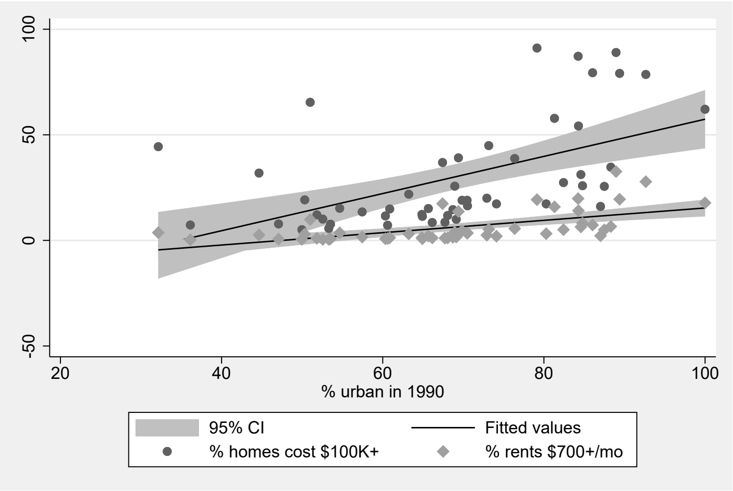
```
tw (lfitci propval100 urban, bcolor(green%20) lcolor(green))
(lfitci rent700 urban, bcolor(orange%20) lcolor(orange)) (scat
propval100 urban, mcolor(%50)) (scat rent700 urban,
mcolor(%50)), pcycle(2) scheme(s1color)
```

This example adds options to improve the appearance of the prior graph. Each `lfitci` command uses the `lcolor()` option to color the fit line to match the markers and the `bcolor()` option to display each confidence region using the color to match the markers but with 20% opacity. Each `scatter` command uses the `mcolor(%50)` option to display the markers with 50% opacity. *Uses allstates.dta*



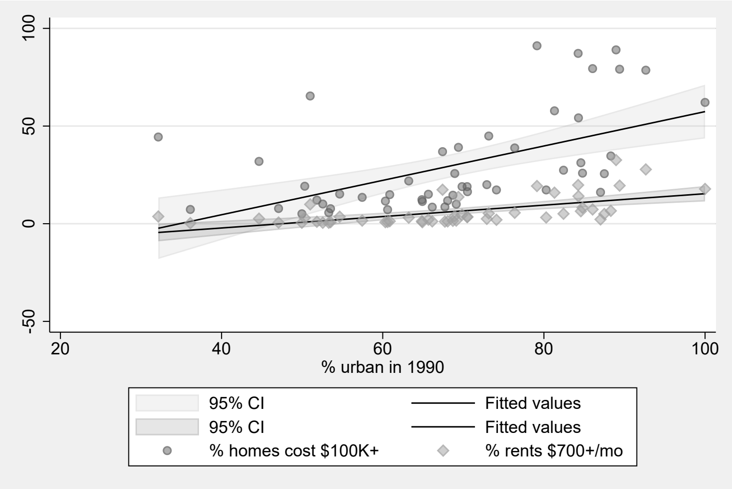
```
twoway (lfitci propval100 urban) (lfitci rent700 urban)
(scatter propval100 urban) (scatter rent700 urban), pcycle(2)
scheme(s2mono)
```

This example illustrates an overlaid scatterplot with linear fit and confidence regions using the `s2mono` scheme. Notice how the confidence regions are the same color and how the overlap is hard to distinguish. Similarly, overlapping markers may conceal one another. *Uses allstates.dta*



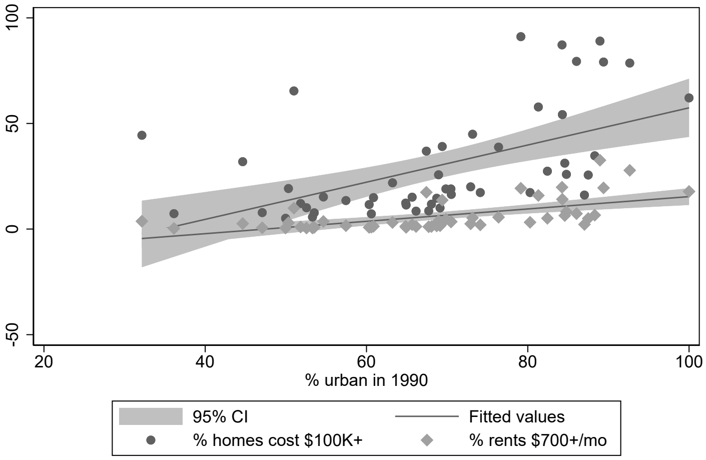
```
tw (lfitci propval100 urban, bcolor(gs9%10)) (lfitci rent700
urban, bcolor(gs1%10)) (scat propval100 urban, mcolor(%50))
(scat rent700 urban, mcolor(%50)), pcycle(2) scheme(s2mono)
```

To improve the prior graph, I added the `bcolor()` option to change the color and opacity of each confidence region, using differing levels of gray (`gs9` versus `gs1`), making the overlap of the confidence intervals more visible. Additionally, I reduced the opacity of the markers with the `mcolor(%50)` option, so the markers do not conceal each other as much. *Uses allstates.dta*



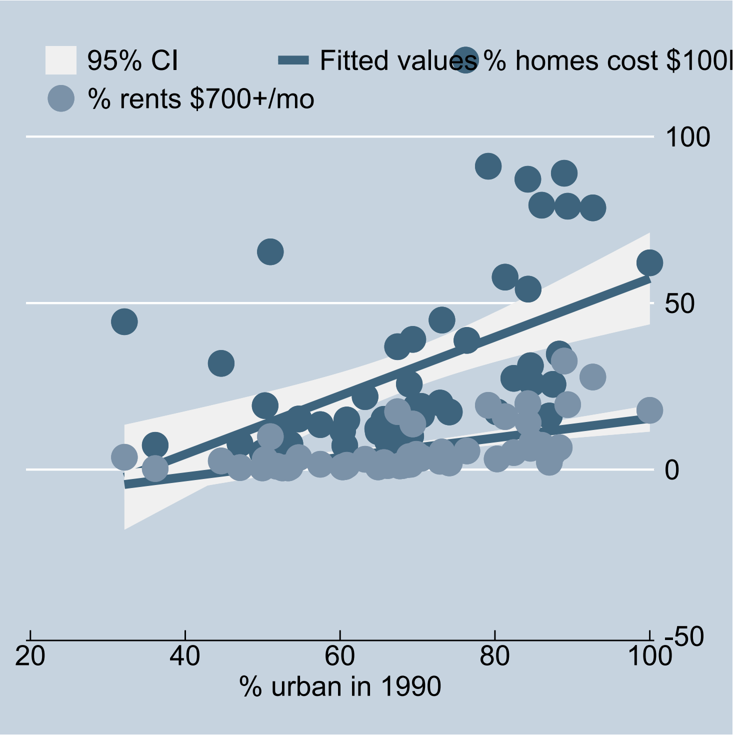
```
twoway (lfitci propval100 urban) (lfitci rent700 urban)
(scatter propval100 urban) (scatter rent700 urban), pcycle(2)
scheme(s1mono)
```

This example uses the `s1mono` scheme and suffers the same kinds of problems as the example using the `s2mono` scheme (that is, overlapping confidence regions and markers potentially concealing one another). I would apply the same options to this graph as I applied to the graph using the `s2mono` scheme, not shown to save space. *Uses allstates.dta*



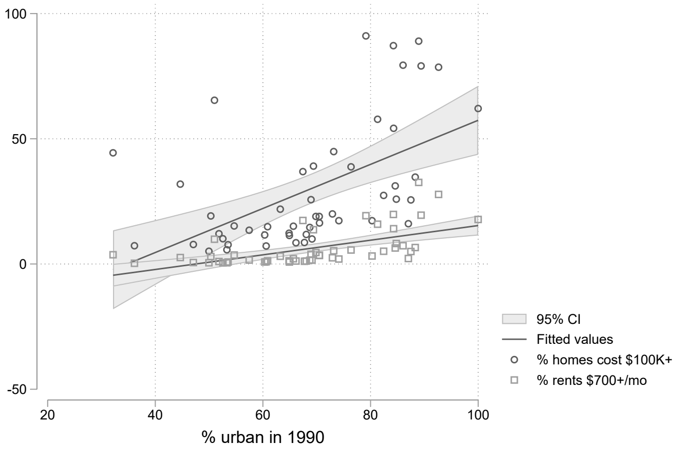
```
twoway (lfitci propval100 urban) (lfitci rent700 urban) (scat
propval100 urban) (scat rent700 urban), pcycle(2)
scheme(economist)
```

This example illustrates an overlaid scatterplot with linear fit and confidence regions using the `economist` scheme. I could apply options to try to improve the appearance of this graph. However, I think the takeaway from this example is that the `economist` scheme is not well suited for this kind of graph, not without extensive use of options to customize its appearance. *Uses allstates.dta*

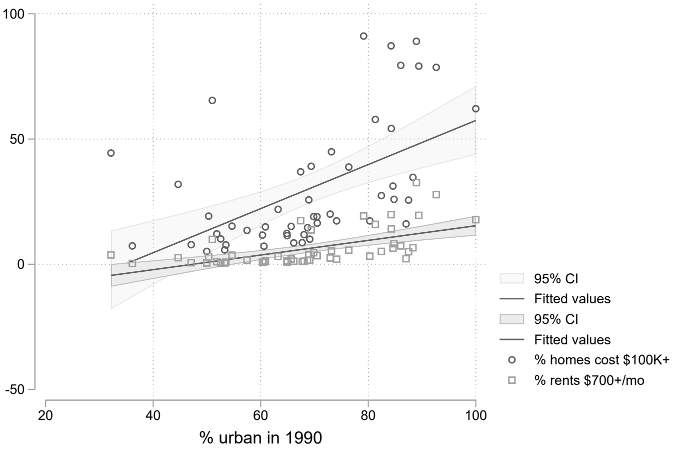


```
twoway (lfitci propval100 urban) (lfitci rent700 urban)
(scatter propval100 urban) (scatter rent700 urban), pcycle(2)
scheme(plotplain)
```

This example illustrates an overlaid scatterplot with linear fit and confidence regions using the `plotplain` scheme. It is possible, although difficult, to discern the overlap in the confidence regions. The style of the markers reduces the chances of overlapping markers concealing one another. By using small hollow markers, we would make two observations that are very close to one another less likely to conceal one another. See section 9.2.2to download this scheme. *Uses allstates.dta*

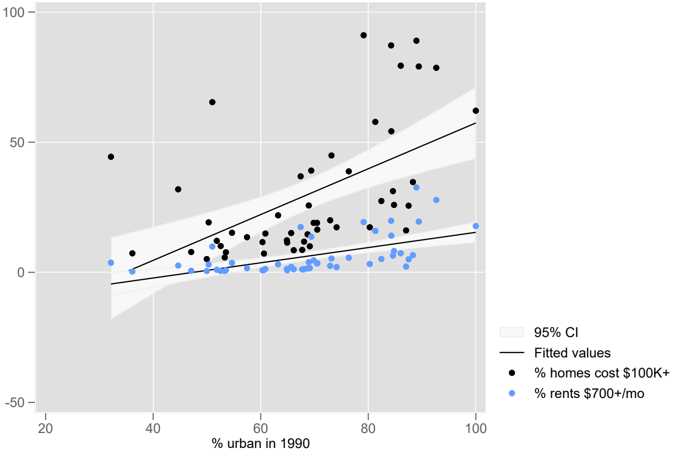


```
twoway (lfitci propval100 urban, bcolor(%30)) (lfitci rent700
urban) (scatter propval100 urban) (scatter rent700 urban),
pcycle(2) scheme(plotplain)
```



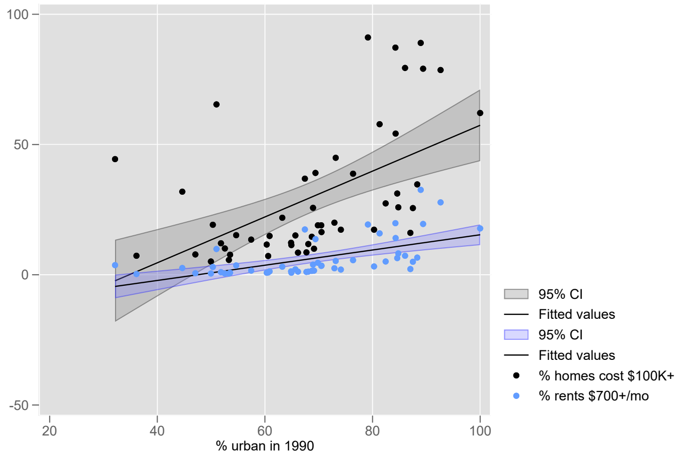
I customized the prior graph by reducing the opacity of the confidence region for home cost, using the `bcolor(%30)` option. The overlap of the confidence regions is easier to discern in this version. The graph created using the `plotplainblind` scheme is similar to this graph, so I am not showing an example using that scheme for this example. *Uses allstates.dta*

```
twoway (lfitci propval100 urban) (lfitci rent700 urban)
(scatter propval100 urban) (scatter rent700 urban), pcycle(2)
scheme(plottig)
```



This example illustrates an overlaid scatterplot with linear fit and confidence regions using the `plottig` scheme. (See section 9.2.2to download this scheme.) The overlapping confidence ranges are hard to distinguish in this graph. I will add options to fix that in the next graph. The plot using the `plottigblind` scheme is similar and will not be shown. *Uses allstates.dta*

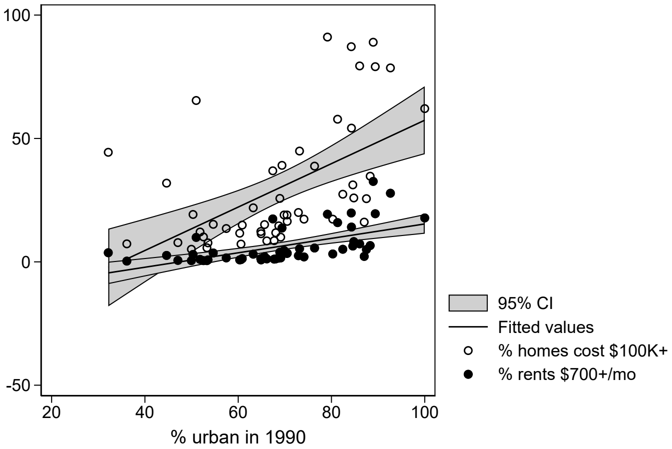
```
twoway (lfitci propval100 urban, bcolor(black%30)) (lfitci
rent700 urban, bcolor(blue%30)) (scatter propval100 urban)
(scatter rent700 urban), pcycle(2) scheme(plottig)
```



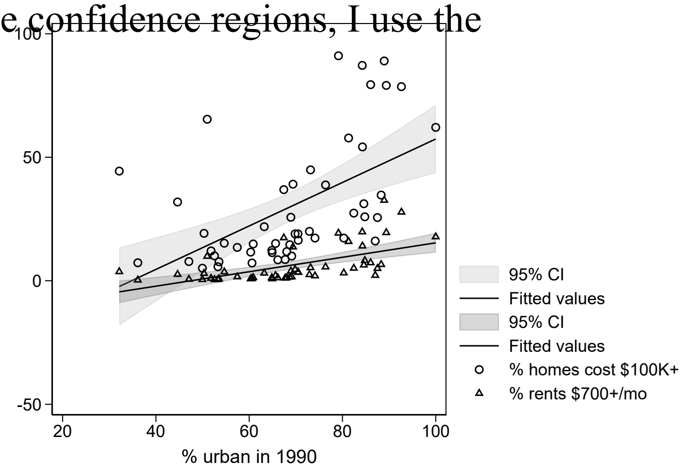
This example changes the color and opacity for each of the confidence regions, making the confidence region color mirror the marker color and clearly displaying the overlap of the confidence regions. The `bcolor(black%30)` option is used for the first `lfitci` command, and the `bcolor(blue%30)` option is used for the second `lfitci` command. *Uses allstates.dta*

```
twoway (lfitci propval100 urban) (lfitci rent700 urban)
(scatter propval100 urban) (scatter rent700 urban), pcycle(2)
scheme(lean1)
```

This example illustrates an overlaid scatterplot with linear fit and confidence regions using the `lean1` scheme. See section 9.2.2to download this scheme. *Uses allstates.dta*

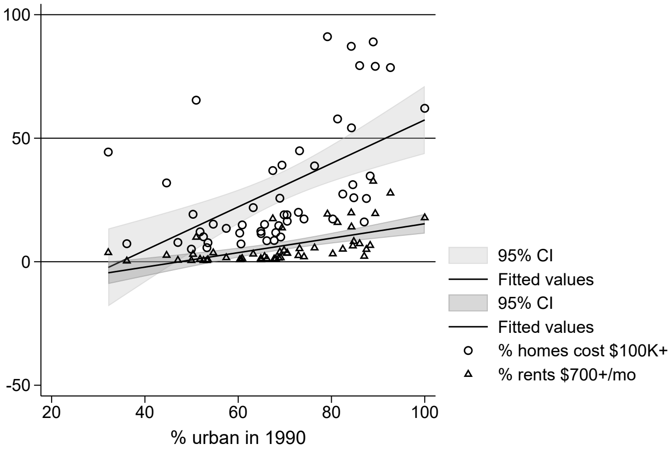


```
twoway (lfitci propval100 urban, bcolor(gs12%30)) (lfitci
rent700 urban, bcolor(gs8%30)) (scatter propval100 urban)
(scatter rent700 urban, msymbol(th)), pcycle(2) scheme(lean1)
```



To better show the overlap of the confidence regions, I use the `bcolor(gs12%30)` and `bcolor(gs8%30)` options to show the home prices with a light-gray confidence region and the rents with a dark-gray confidence region, both displayed with 30% opacity. Note hollow triangles were used to display the markers for rents to try to minimize concealment of markers due to overlapping. *Uses allstates.dta*

```
twoway (lfitci propval100 urban, bcolor(gs12%30)) (lfitci
rent700 urban, bcolor(gs8%30)) (scatter propval100 urban)
(scatter rent700 urban, msymbol(th)), pcycle(2) scheme(lean2)
```



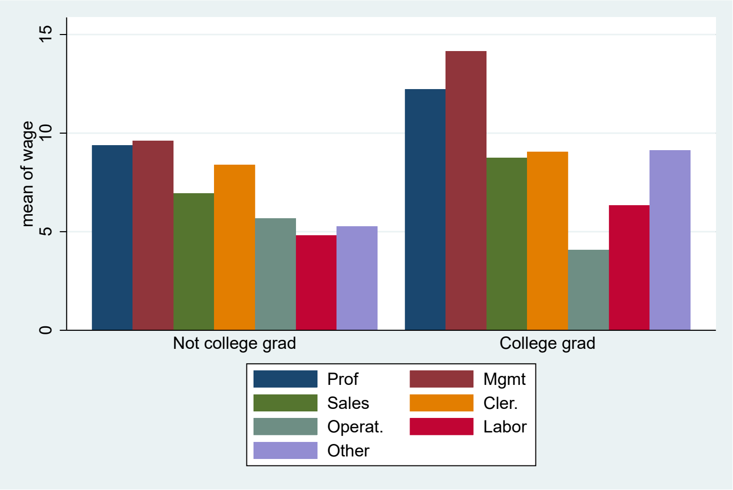
This example is identical to the prior example but now uses the `lean2` scheme. Note this example has applied the `bcolor(gs12%30)` option to make the confidence region for the home costs light gray with 30% opacity. I also used the `bcolor(gs8%30)` option to make the confidence region for the rents a darker gray with 30% opacity. Also, I added `msymbol(th)` to display the rents using small hollow triangles. See section 9.2.2to download this scheme. *Uses allstates.dta*

Example #3: A bar chart

This example shows a bar chart with two `over()` variables. The first `over()` variable is occupation (with seven levels) and is displayed with different colors using the `asyvars` option. In this example, I will particularly focus on the colors used by each of the schemes when displaying the different bars and the legend that labels the bars but will still note other key characteristics of the graphs.

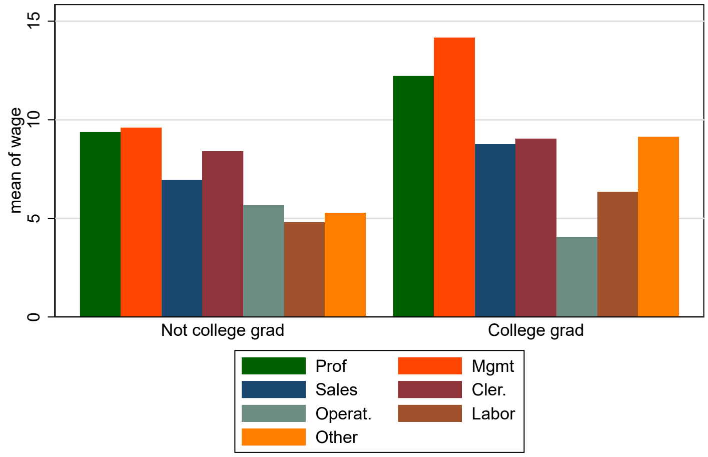
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s2color)
```

This shows an example of a bar chart using the `s2color` scheme. Note that the bars are displayed using different colors. The plot region has a white background, while the graph region has a light-blue background. The legend is displayed at the bottom of the graph, and the y-axis labels are rotated 90 degrees. *Uses nlsw.dta*



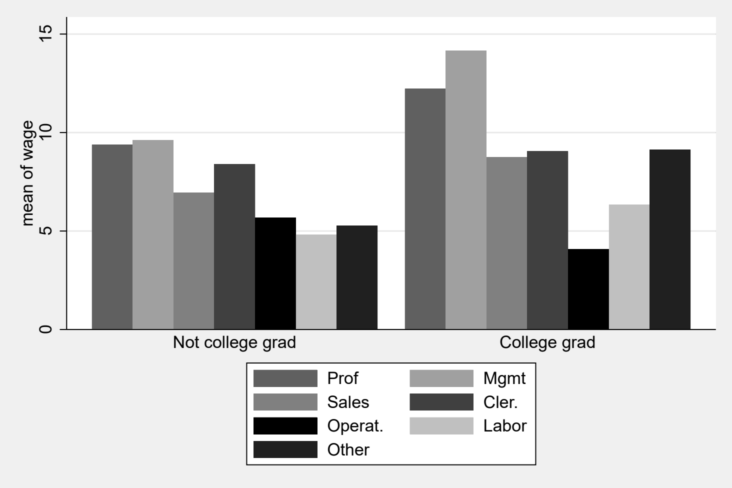
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s1color)
```

This shows an example of a bar chart using the `s1color` scheme. The bars are displayed using different colors and colors different from those used with the `s2color` scheme. The entire graph has a white background. The legend is displayed at the bottom of the graph, and the y-axis labels are rotated 90 degrees. *Uses nlsw.dta*



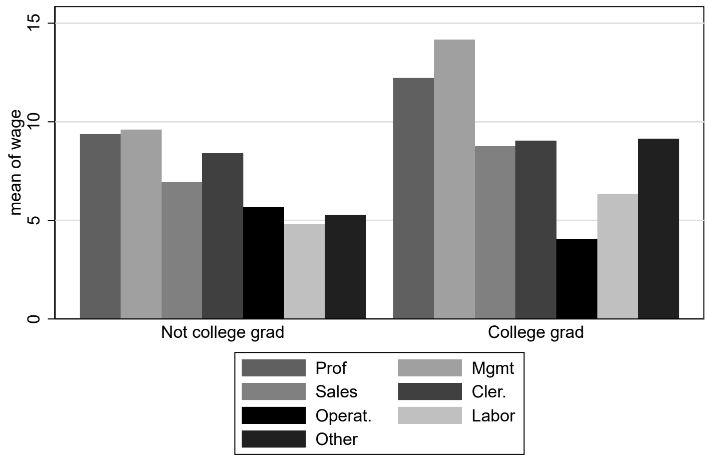
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s2mono)
```

This shows an example of a bar chart using the `s2mono` scheme. The bars are displayed using different shades of gray. The plot region has a white background, while the graph region has a light-gray background. The legend is displayed at the bottom of the graph, and the y-axis labels are rotated 90 degrees. *Uses nlsw.dta*



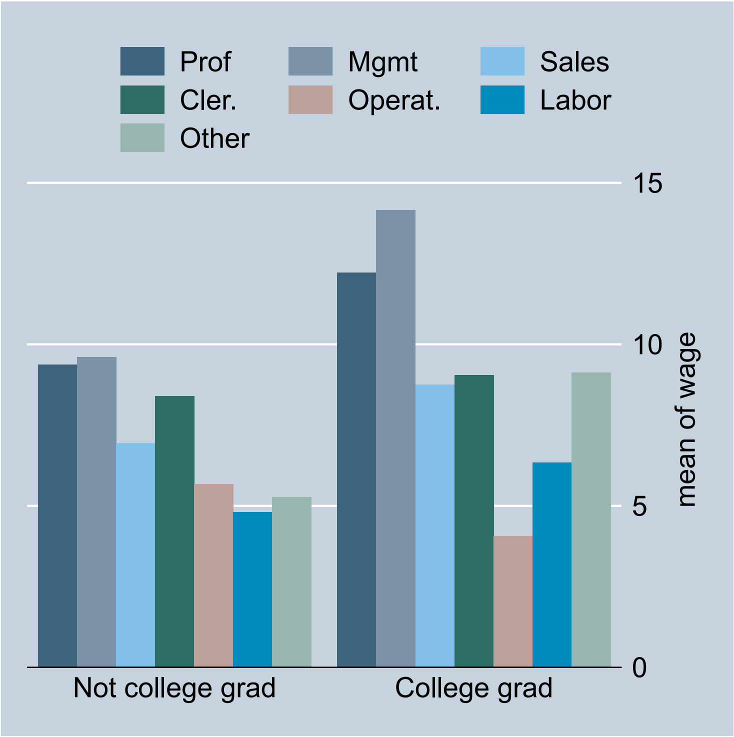
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(s1mono)
```

This shows an example of a bar chart using the `s1mono` scheme. The bars are displayed using different shades of gray, the same shades of gray as used in the `s2mono` scheme. The entire graph background is white. The legend is displayed at the bottom of the graph, and the y-axis labels are rotated 90 degrees. *Uses nlsw.dta*



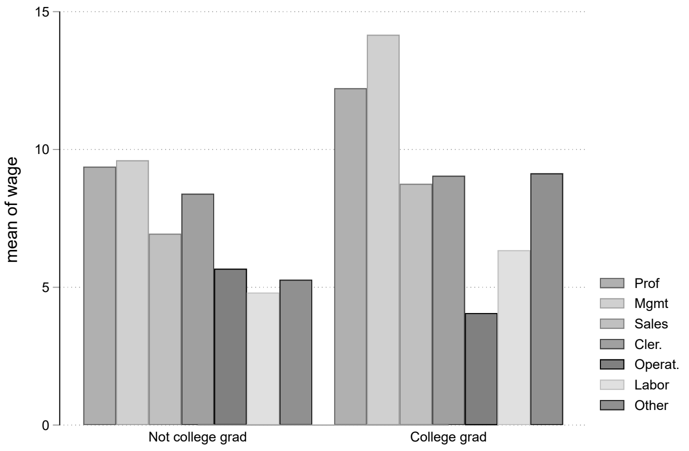
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(economist)
```

This shows an example of a bar chart using the `economist` scheme. The entire graph is shown using a grayish background color. The bars are different colors, with the legend displayed at the top. The y-axis labels are displayed on the right side of the plot region and are not rotated. *Uses nlsw.dta*



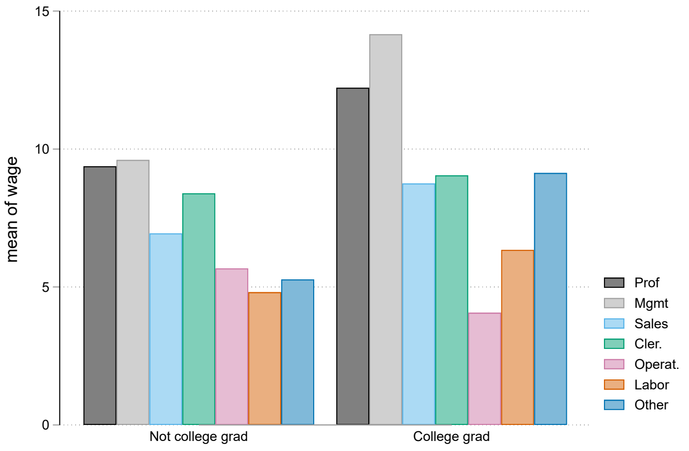
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(plotplain)
```

This shows an example of a bar chart using the `plotplain` scheme. The bars are shown using different shades of gray, and the entire graph background is white. The legend is compactly displayed to the right of the plot region, resulting in a plot region that is more square than the prior examples (which were wider and shorter). *Uses nlsw.dta*



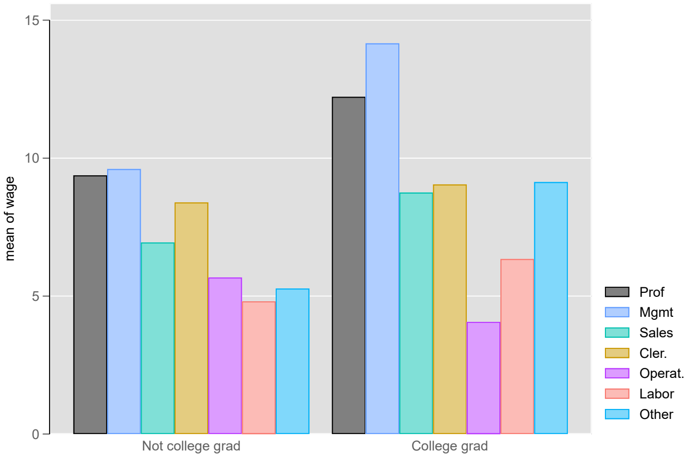
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(plotplainblind)
```

This shows an example of a bar chart using the `plotplainblind` scheme. The bars are shown using different colors, including two shades of gray, that are more readable to those with color blindness. The entire background is white. As with the prior example, the legend is compactly displayed to the right of the plot region, and the plot region is more square than the examples using the `s1` or `s2` scheme family. *Uses nlsw.dta*

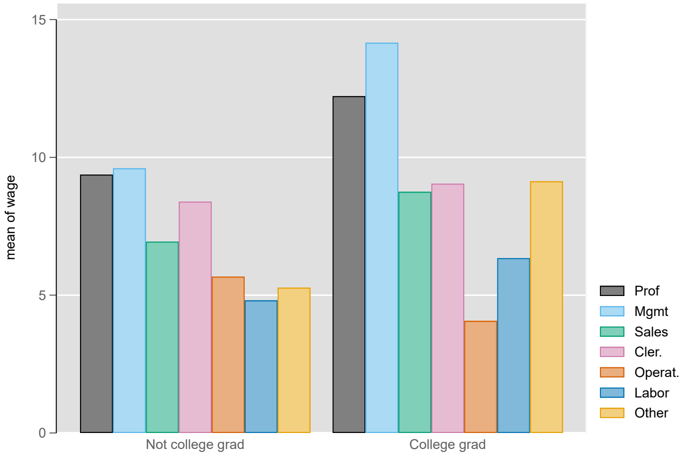


```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(plottig)
```

This shows an example of a bar chart using the `plottig` scheme. The bars are shown using different colors. The plot region is gray and the graph region is white. As with the prior example, the legend is compactly displayed to the right of the plot region. *Uses nlsw.dta*



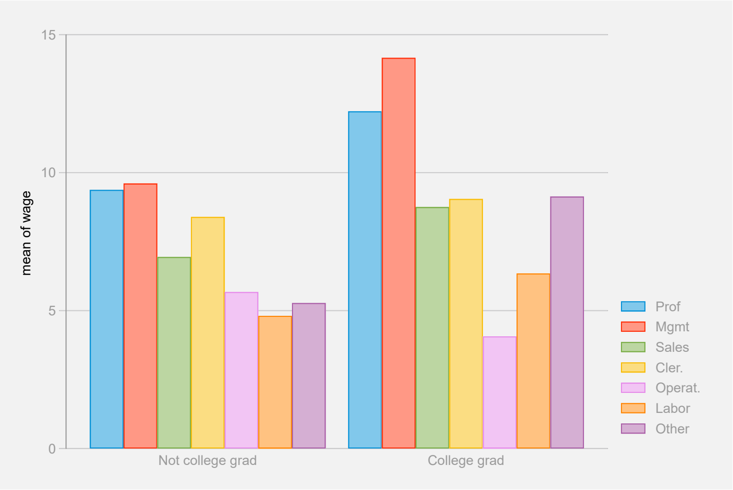
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(plottigblind)
```



This shows an example of a bar chart using the `plottigblind` scheme. The appearance of this scheme is very similar to the prior example. The key difference is that the colors are (somewhat) different, using colors that are more readable to those with color blindness. *Uses nlsw.dta*

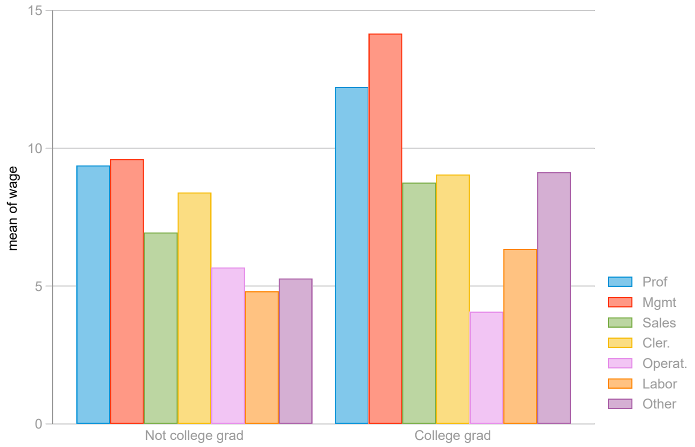
```
graph bar wage, over(occ7) over(collgrad) asyvars scheme(538)
```

This shows an example of a bar chart using the `538` scheme. This entire graph has a gray background, and the bars are shown using different colors. As with the prior examples, the legend is compactly displayed to the right of the plot region, and the plot region is more square than the examples using the `s1` or `s2` family of schemes. *Uses nlsw.dta*



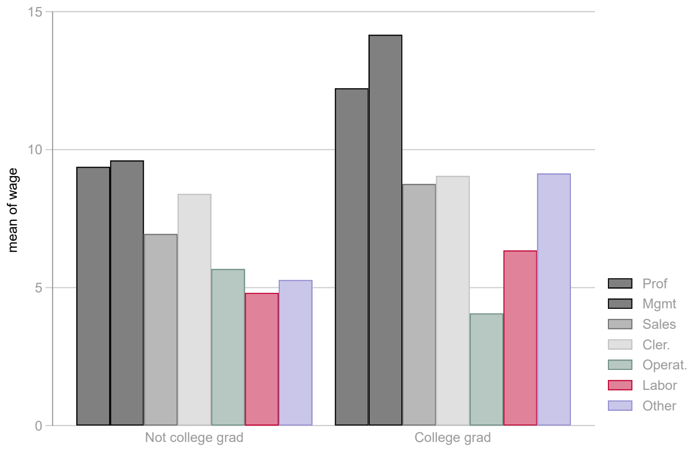
```
graph bar wage, over(occ7) over(collgrad) asyvars scheme(538w)
```

This shows an example of a bar chart using the `538w` scheme. As you can see, this graph is identical to the prior graph, except that the entire background is shown in white. *Uses nlsw.dta*

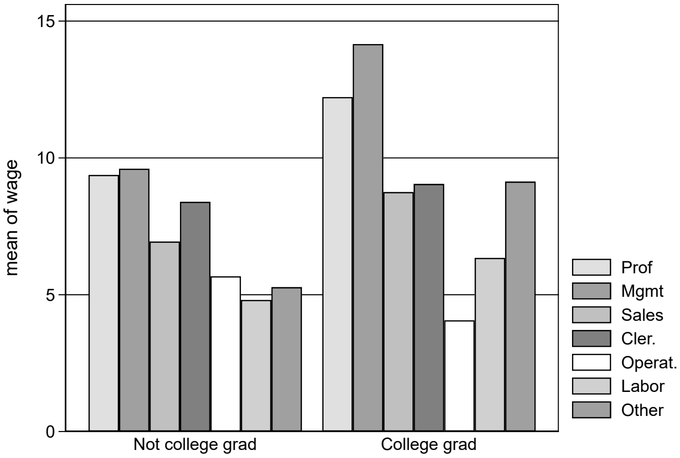


```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(538bw)
```

This shows an example of a bar chart using the `538bw` scheme. This graph looks very much like the prior graph, except for the color of the bars. I expected the bars to be black and white, but the first two bars are very similar in color, and the last three bars are colors. This graph would require additional options to produce a black-and-white version, in which case `s1mono`, `plotplain`, `lean1`, or `lean2` might be a better alternative. *Uses nlsw.dta*



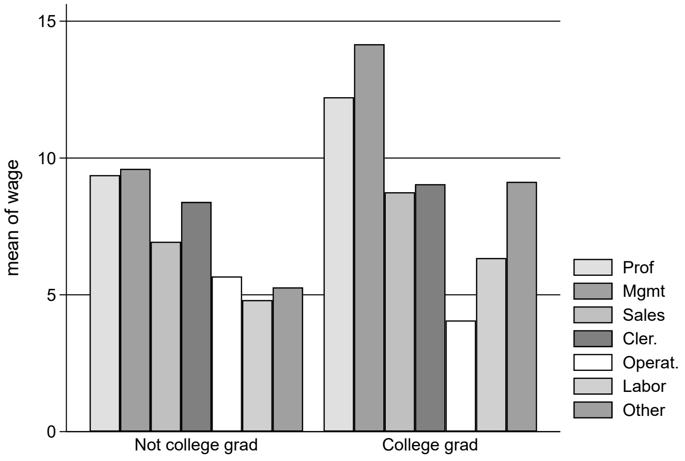
```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(lean1)
```



This shows an example of a bar chart using the `lean1` scheme. The bars are shown using different shades of gray, and the entire graph has a white background. The plot area is framed, and the legend is displayed to the right of the plot area. The y-axis labels are not rotated. *Uses nlsw.dta*

```
graph bar wage, over(occ7) over(collgrad) asyvars
scheme(lean2)
```

This shows an example of a bar chart using the `lean2` scheme. This example is nearly identical to the previous example, except that the plot area is not framed. *Uses nlsw.dta*



The grstyle way of customizing graphs

Ben Jann has another take on customizing graphs, as described in two articles published in the Stata Journal; see

- Jann, B. 2018. Customizing Stata graphs made easy (part 1). *Stata Journal* 18: 491–502. https://doi.org/10.1177/1536867X1801800301.
- Jann, B. 2018. Customizing Stata graphs made easy (part 2). *Stata Journal* 18: 786–802. https://doi.org/10.1177/1536867X1801800403.

9.2.6 Customizing schemes

This section shows how to customize your own schemes. Although schemes can look complicated, it is possible to easily create some simple schemes on your own. Let’s look at the `vg_lgndc` scheme as an example. This scheme is based on the `s2color` scheme but changes the legend to display at the nine o’clock position, in one column, with the keys stacked on top of the symbols. Here are the contents of that scheme:

```
#include s2color // start with the s2color scheme
clockdir legend_position  9   // put the legend in the nine o´clock position
numstyle legend_cols      1   // make the legend display in 1 column
yesno legend_stacked      yes // stack the keys & symbols on top of each other
gsize legend_key_gap    half_tiny // very, very small gap between key and label
gsize legend_row_gap    small     // somewhat larger gap between key/label pairs
```

Rather than creating the `vg_lgndc` scheme from scratch, which would be laborious, I used the `#include s2color` statement to base this new scheme on the `s2color` scheme. The subsequent statements changed the position of the legend and the number of columns in the legend and stacked the legend keys and symbols upon each other.

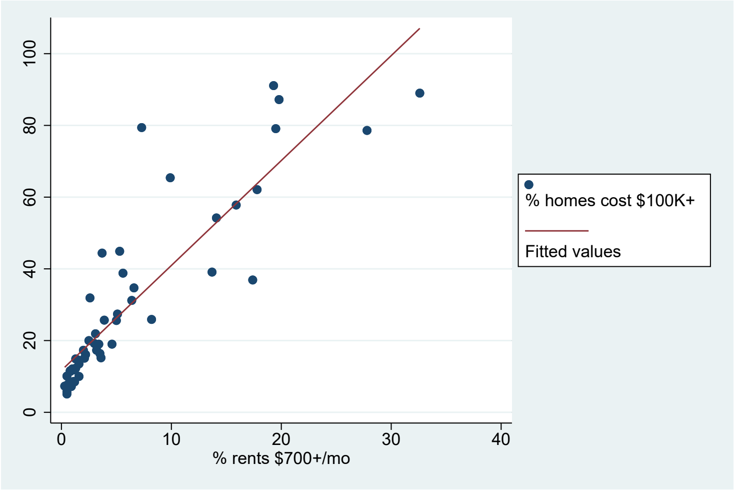
Say that we liked the `vg_lgndc` scheme but wanted to make our own version in which the legend is in the three o'clock position instead of the nine o'clock position, naming our version `legend3`. To do this, we would start the Stata Do-file Editor, for example, by typing `doedit`, and then type the following into it: (Of course, the scheme will work fine if you omit comments after the double slashes.)

```
#include s2color // start with the s2color scheme
clockdir legend_position  3   // put the legend in the three o´clock position
numstyle legend_cols      1   // make the legend display in 1 column
yesno legend_stacked      yes // stack the keys & symbols on top of each other
gsize legend_key_gap    half_tiny // very, very small gap between key and label
gsize legend_row_gap    small     // somewhat larger gap between key/label pairs
```

We can then save the file as `scheme-legend3.scheme`. We can use the `scheme(legend3)` option at the end of a graph command or type `set scheme legend3`, and Stata will use that scheme for displaying our graph. Below we show an example using this scheme. (The `legend3` scheme is not included among the downloadable schemes.)

```
twoway (scatter propval100 rent700) (lfit propval100 rent700),
scheme(legend3)
```

This is an example using the newly created `legend3` scheme. We see the legend in the three o'clock position, in one column, with the legend stacked.
Uses allstates.dta



So far, things are going great. However, Stata will only know how to find the newly created `scheme-legend3.scheme` while you are working in the directory where you saved that scheme. If you change to a different directory, Stata will not know where to find `scheme-legend3.scheme`. If, however, we save the scheme into your `PERSONAL` directory, Stata will know where to find it regardless of your current directory. For example, on my computer, I typed the `sysdir` command, which produced

```
. sysdir
   STATA:  C:\Program Files (x86)\Stata17\
 UPDATES:  C:\Program Files (x86)\Stata17\ado\updates\
    BASE:  C:\Program Files (x86)\Stata17\ado\base\
    SITE:  C:\Program Files (x86)\Stata17\ado\site\
    PLUS:  c:\ado\plus\
PERSONAL:  c:\ado\personal\
OLDPLACE:  c:\ado\
```

From this, I see that my `PERSONAL` directory is located in `c:\ado\personal\`, so if I store either `.ado` files or `.scheme` files there, Stata will find them. So, instead of saving `scheme-legend3.scheme` into the current directory, you may want to save it into your `PERSONAL` directory. You may need to create that folder/directory if it does not already exist. (Note that if you save `scheme-legend3.scheme` to the current directory and also save it to the `PERSONAL` directory, you should remove the copy from the current directory.)

This section has really focused on the very basics of creating a scheme. Stata has greatly enhanced its documentation about how to create scheme files of your own. Type `help scheme files` for more information on creating your own schemes.

9.2.7 Using the set scheme command

The previous sections have illustrated a number of different schemes. As these examples have shown, we can change the scheme of a graph by supplying the `scheme()` option on a graph command. If you want to use the same scheme over and over, you can use the `set scheme` command to set the default scheme. For example, if you typed

```
. set scheme economist
```

the default scheme would become `economist` until you quit Stata. Or you could type

```
. set scheme economist, permanently
```

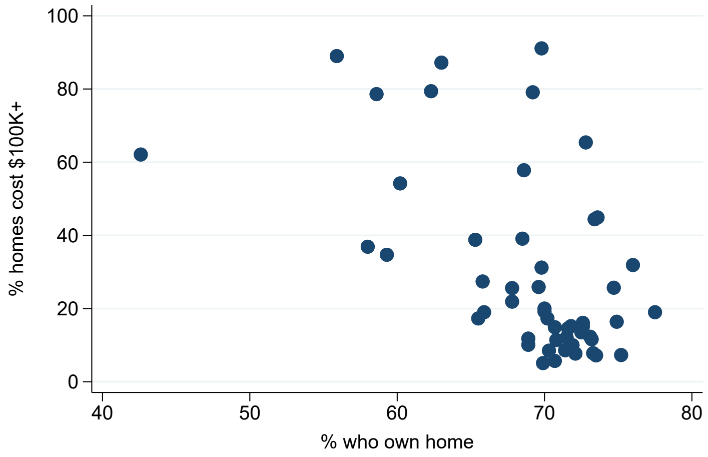
and the `economist` scheme would be your default scheme, even after you quit and restart Stata. If you will be creating a series of graphs that you want to have a common look, then using the `set scheme` command can be a great shortcut to save you the effort of needing to specify the same scheme over and over via the `scheme()` option.

9.3 Sizing graphs and their elements

This section illustrates how to use the `xsize()` and `ysize()` options to specify the size of graphs, using points, inches, or centimeters. Using these options, you can create very small graphs (as small as 1 inch by 1 inch) and very large graphs (as large as 100 inches by 100 inches). This section also illustrates the `aspectratio()` option to control the aspect ratio of the plot region, as well as the `scale()` option to control the size of the text and markers. Additionally, in section 9.3.1, I illustrate what happens when you change the size of graphs (with the `xsize()` and `ysize()` options) when the graphs contain elements that were sized using absolute units (that is, points, inches, centimeters) versus when elements were sized using relative units (for example, `size(large)`, `size(*2)`, or `size(5rs)`).

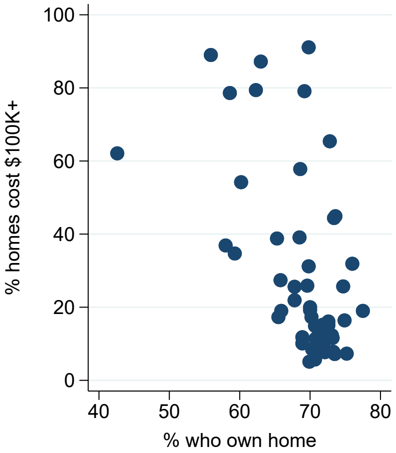
```
scatter propval100 ownhome
```

Let’s first consider this graph. The graphs in this book have been sized to be 3 inches wide by 2 inches tall. *Uses allstates.dta*

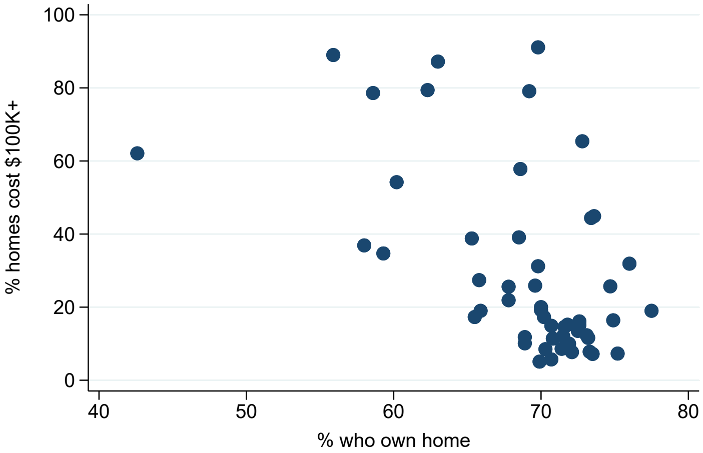


```
scatter propval100 ownhome, aspectratio(1.3)
```

You can use the `aspectratio()` option to change the aspect ratio of the plot region of the graph. Numbers greater than 1 create tall skinny plot regions, and numbers less than 1 create wide fat plot regions. *Uses allstates.dta*



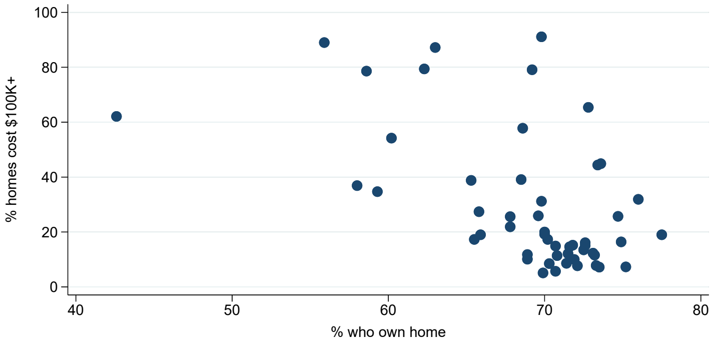
```
scatter propval100 ownhome, xsize(6cm) ysize(4cm)
```



The `xsize()` and `ysize()` options allow us to set the size of these dimensions using inches, centimeters, or points. If we do not explicitly specify the units of size, the `xsize()` and `ysize()` options assume we are specifying inches. In this example, we specifically indicate that we want the graph to be 6 centimeters wide and 4 centimeters high. *Uses allstates.dta*

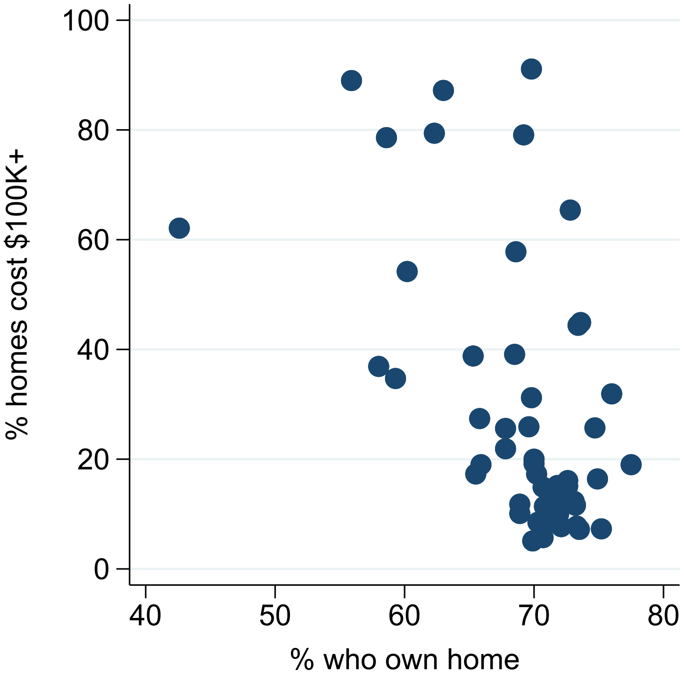
```
scatter propval100 ownhome, xsize(4in) ysize(2in)
```

Here we use `xsize(4in)` and `ysize(2in)` to make the graph 4 inches wide and 2 inches tall. *Uses allstates.dta*



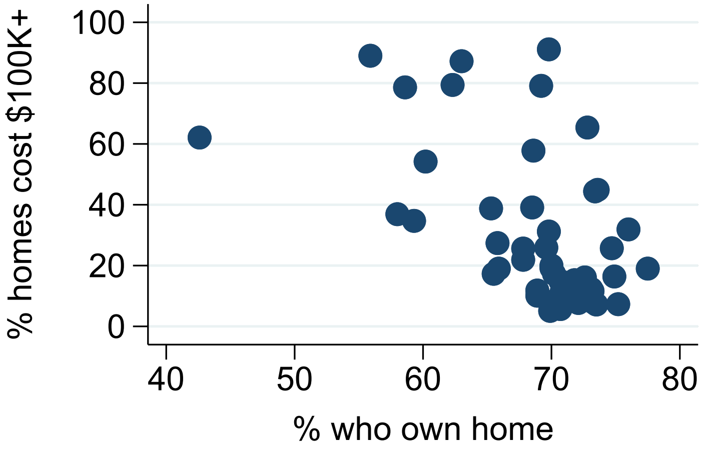
```
scatter propval100 ownhome, xsize(2in) ysize(2in)
```

Here is another example illustrating the `xsize()` and `ysize()` options. In this example, we use these options to create a graph 2 inches wide by 2 inches tall. *Uses allstates.dta*



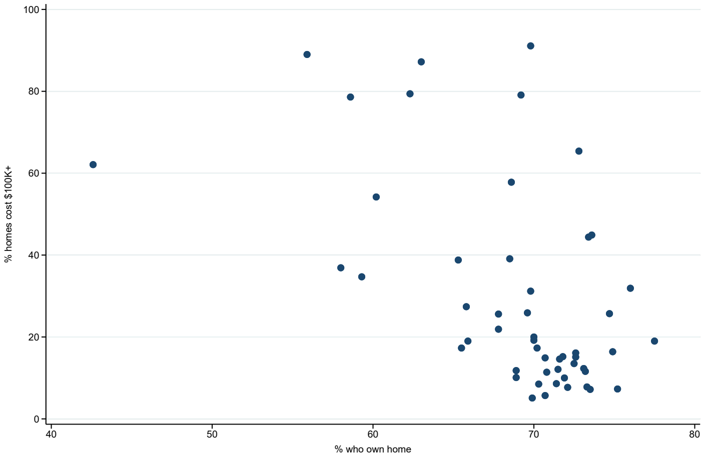
```
scatter propval100 ownhome, scale(1.7)
```

Here we add the `scale(1.7)` option to magnify the sizes of the text and markers in the graph, making them 1.7 times their normal sizes. This option can be useful when creating small graphs and increasing the sizes of the text and markers would make them easier to see. *Uses allstates.dta*



```
scatter propval100 ownhome, scale(.5)
```

We can also use the `scale()` option to decrease the sizes of the text and markers. Here we make these elements half their normal size. *Uses allstates.dta*

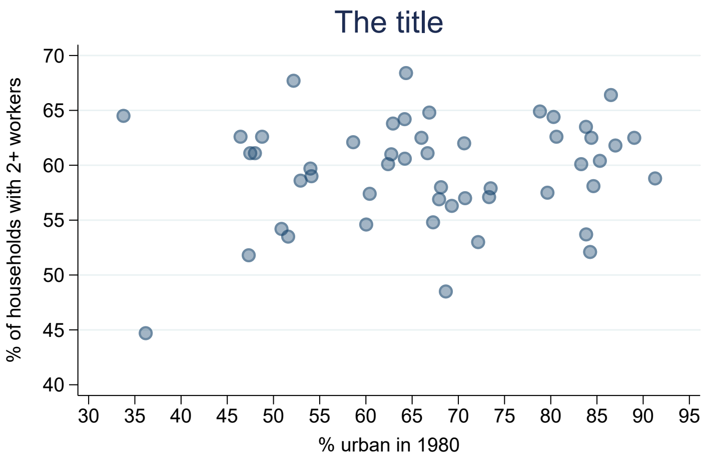


9.3.1 Sizing/resizing graphs with absolutely sized versus relatively sized units

Throughout this book, I have illustrated how to control the size of graphical elements like markers, titles, subtitles, axis titles, axis labels, makers, and so forth. I have consistently demarcated methods that control size using *absolute units* (like points, inches, and centimeters) versus methods that use *relative units* (for example, `size(large)`, `size(*2)`, or `size(10rs)`). For the sake of an example, consider this graph below.

```
scatter workers2 pcturban80, mcolor(%40) title("The title")
xlabel(30(5)95, format(%2.0f)) ylabel(40(5)70)
```

This is a scatterplot showing, at the state level, the percentage of households with 2+ workers by the percentage living in urban areas. *Uses allstates.dta*



I am delighted to say that I showed this graph to an editor from the fictitious *Journal of We Publish Anything*. She wants to publish my article using this figure. The only stipulation is that my graph needs to conform to the journal's strict sizing guidelines. I have listed its guidelines along with a reference to the part of the book that shows how to implement that specification.

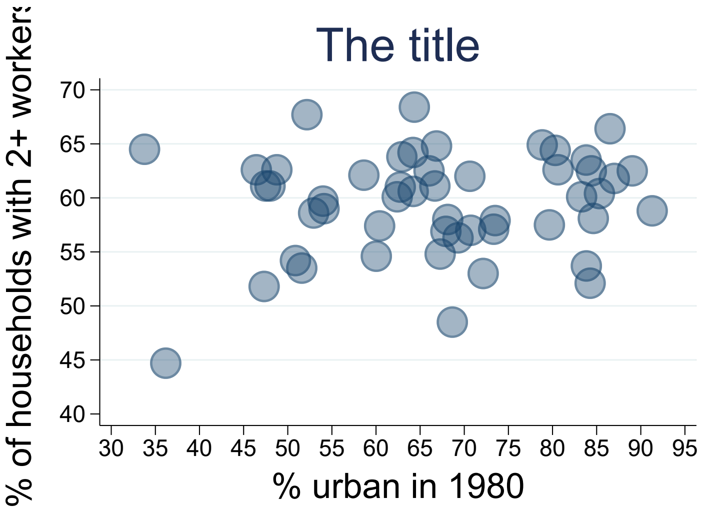
1. The marker sizes must be 10 points [see Styles : Markersize (section 10.9)].
2. The main title must be 16 points [see Styles : Textsize (section 10.12)].
3. The axis titles must be exactly 12 points [see Styles : Textsize (section 10.12)].
4. The axis labels must be exactly 8 points [see Styles : Textsize (section 10.12)].
5. The graph must be 3.5 inches wide and 2 inches tall [see Standard options : Sizing graphs (section 9.3)].

Using the resources noted above, I show a graph that reflects these specifications.

```
scatter workers2 pcturban80, xsize(3.5in) ysize(2.5in)
title("The title", size(16pt)) msize(10pt) mcolor(%40)
```

```
xtitle( , size(12pt)) xlabel(30(5)95, labsize(8pt)
format(%2.0f)) ytitle( , size(12pt)) ylabel(40(5)70,
labsize(8pt))
```

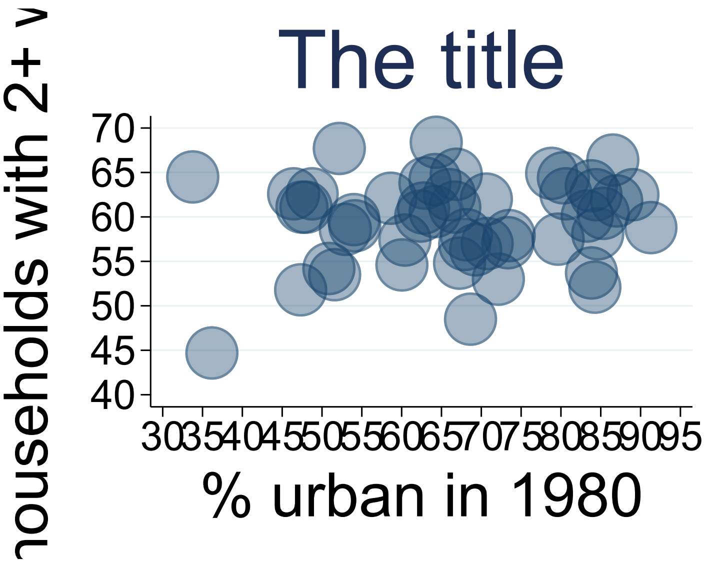
This graph uses markers that are 10 points; the title is 16 points; the axis titles are each 12 points; the axis labels are each 8 points. The overall size of the graph is 3.5 inches wide and 2.5 inches high. *Uses allstates.dta*



The sad news is that the *Journal of We Publish Anything* went out of business before my article was published. The silver lining is that the fictitious *Journal of Tiny Figures* is very interested in my article. Its figure guidelines are far more flexible, with the only stipulation being that the figure should be well labeled, and it must be 2 inches wide and 1.5 inches tall. With glee and optimism, I modify my `scatter` command from above, adding the new size specifications of `xsize(2in) ysize(1.5in)`.

```
scatter workers2 pcturban80, xsize(2in) ysize(1.5in)
title("The title", size(16pt)) msize(10pt) mcolor(%40)
xtitle( , size(12pt)) xlabel(30(5)95, labsize(8pt)
format(%2.0f)) ytitle( , size(12pt)) ylabel(40(5)70,
labsize(8pt))
```

I took the graph that I created for the *Journal of We Publish Anything* and changed the size to accommodate the requirements of the *Journal of Tiny Figures*. Namely, I made the graph 2 inches wide and 1.5 inches tall by specifying `xsize(2in) ysize(1.5in)`. This graph does not look very good. *Uses allstates.dta*



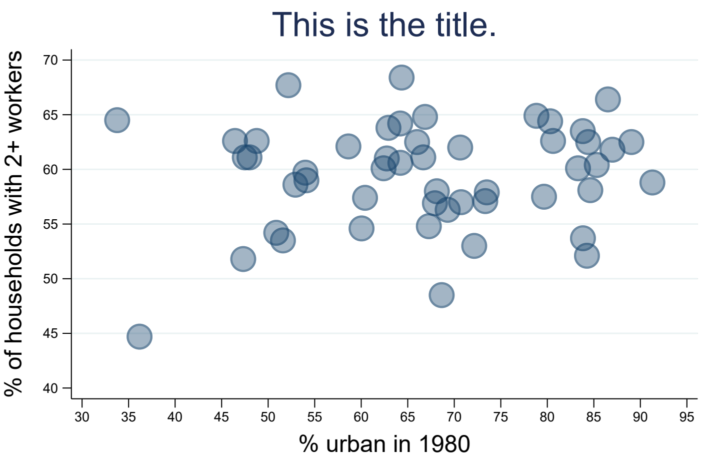
Being in a hurry, I just submitted the manuscript, my Stata commands, and my crummy figure to the *Journal of Tiny Figures*. Fortunately, it turns out the editor is a Stata enthusiast and has recently had many other people, like me, peddling their graphs using the specifications from the failed *Journal of We Publish Anything*. She explained that using their specifications, the size of the graph elements and the overall size of the graphs worked together, in synchrony. They first chose their graph size as 3.5 by 2 inches, and then they chose the size of all the graph elements, specified in points (72nds of an inch), to precisely integrate with the overall graph size. When I tried to resize the graph using the `xsize()` and `ysize()` options, it was like trying to shove a king-sized bed into a kid-sized room. The title, sized using 16 points, looks great on a graph that is 3.5 by 2 inches but looks absurd on a graph that is 2 by 1.5 inches. She took pity on me and gave me a revise and resubmit, including a patient explanation of my options for improving the figure.

Option 1: Resize the elements using absolute units. I could change the size of the title, using points that would be commensurate with an overall graph size of 2 by 1.5 inches. But, if they reject my submission, my next choice is the *Journal of Oversized Figures*, which requires figures that are 20 by 10 inches. I would then have to manually resize the title and all the other elements to match that overall graph size.

Option 2: Resize the elements using relative units. She recommended this option. I would size the elements of the graph, title, axis titles, axis labels, and markers, using relative units (for example, `size(large)`). Then, when I use the `xsize()` and `ysize()` options, all of these elements will be resized proportionately. Referring to Styles : Textsize (section 10.12), I sized the title using `size(huge)`, the axis titles using `size(large)`, the axis labels using `labsize(small)`. Referring to Styles : Markersize (section 10.9), I sized the markers using `msize(huge)`. The resulting graph is shown below.

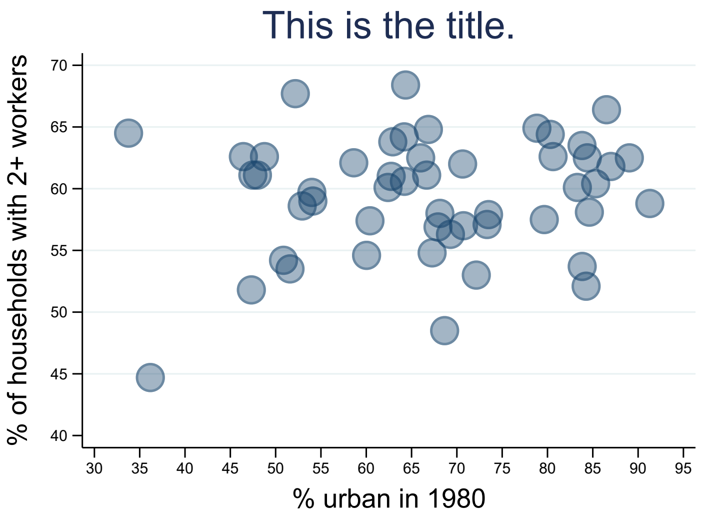
```
scatter workers2 pcturban80, title("This is the title.",
size(huge)) msize(huge) mcolor(%40) xtitle( , size(large))
xlabel(30(5)95, labsize(small) format(%2.0f)) ytitle( ,
size(large)) ylabel(40(5)70, labsize(small))
```

This is my figure, sized using relative units—sizing the title using `size(huge)`, the markers using `msize(huge`, the axis titles using `size(large)`, and the axis labels using `labsize(small)`. In the next example, I will use the `xsize(2)` and `ysize(1.5)` options to size the graph for the *Journal of Tiny Figures*.
Uses allstates.dta



```
scatter workers2 pcturban80, xsize(2in) ysize(1.5in)
title("This is the title.", size(huge)) msize(huge)
mcolor(%40) xtitle( , size(large)) xlabel(30(5)95,
labsize(small) format(%2.0f)) ytitle( , size(large))
ylabel(40(5)70, labsize(small))
```

Here is the same figure but now resized to meet the requirements of the *Journal of Tiny Figures*, using the `xsize(2)` and `ysize(1.5)` options. The figure is smaller, but all the elements have been resized proportionately.
Uses allstates.dta

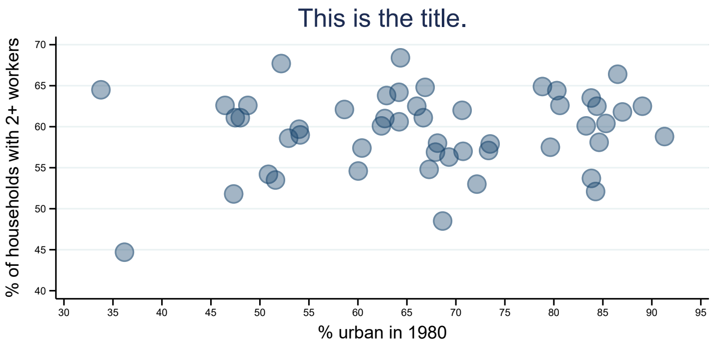


Even though that story was fictitious, I think the lesson from the story is useful. In my work as a statistician, I am frequently asked to create graphs that will be used in multiple ways. A common scenario is that the graph will be used for a publication (sized around 4 inches high) and a large version will be needed for a poster. The size of the poster version will be decided at the last minute based on whatever blank space is left on the poster—perhaps a space that is tall and skinny, or maybe wide and short, or maybe square. For such graphs, I often use my *graph resizing torture test*, where I try displaying the graph in each of these shapes via the `xsize()` and `ysize()` options and checking that the graph will look okay using different dimensions.

Let's put the last figure through the *resizing torture test*, adapted for the size constraints of this book. Let's display this graph 1) 2 inches wide by 1 inch tall, 2) 1 inch wide by 2 inches tall, and 3) 2 inches by 2 inches.

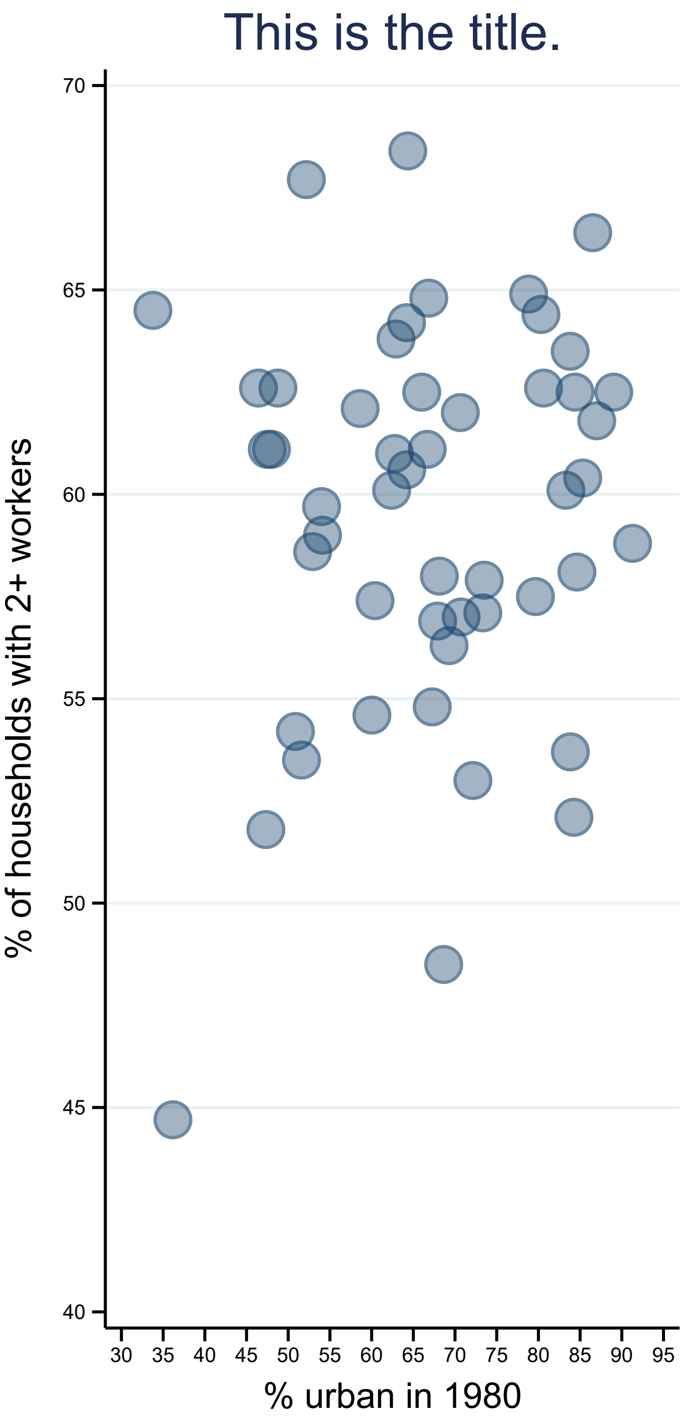
```
graph display, xsize(2in) ysize(1in)
```

The `graph display` command can be used to change the size of the graph currently in memory. By specifying `xsize(2in) ysize(1in)`, we size the graph 2 inches wide and 1 inch tall. Despite changing the size of the graph so much, the overall graph looks good. In particular, I like the sizing of the title, axis titles, axis labels, and markers. *Uses allstates.dta*



```
graph display, xsize(1in) ysize(2in)
```

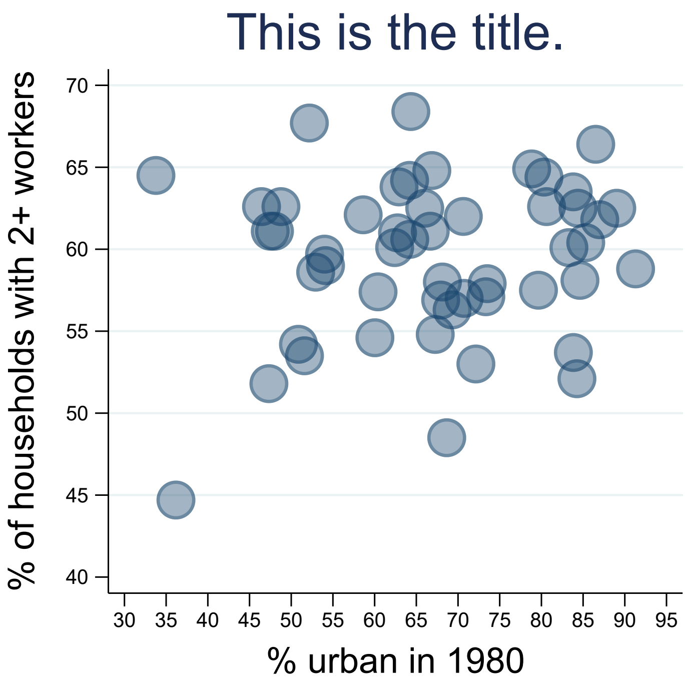
Here is the graph, displayed using torture test #2, using the `xsize()` and `ysize()` options to size the graph as 1 inch wide and 2 inches high. Using this sizing, we see the elements of the graph look good, namely, the title, axis titles, axis labels, and markers. *Uses allstates.dta*



```
graph display, xsize(2in) ysize(2in)
```

Here is the graph, displayed using torture test #3, using the `xsize()` and `ysize()` options to size the graph as 2 inches wide and 2 inches high.

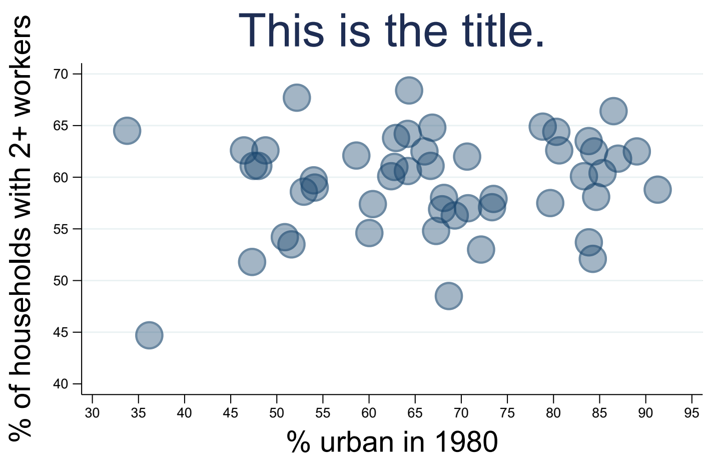
Again, the overall graph looks good, as do the title, axis titles, axis labels, and markers. *Uses allstates.dta*



Referring to **Styles : Textsize** (section 10.12) and **Styles : Markersize** (section 10.9), I could have sized the graph elements using multipliers of the original size. I used that technique below, where I made the markers *2.2 times their normal size, the title *1.5 times the normal size, and the titles of the axes *1.5 times their normal size and made the axis labels use the multiplier *0.7, making them 70% of their normal size.

```
scatter workers2 pcturban80, title("This is the title.",
size(*1.5)) msize(*2.2) mcolor(%40) xtitle( , size(*1.5))
xlabel(30(5)95, labsize(*0.7) format(%2.0f)) ytitle( ,
size(*1.5)) ylabel(40(5)70, labsize(*0.7))
```

Here is my revised figure using multipliers for sizing. *Uses allstates.dta*



We could then put this version through the *graph resizing torture test* using the commands below.

```
 graph display, xsize(2in) ysize(1in)
graph display, xsize(1in) ysize(2in)
graph display, xsize(2in) ysize(2in)
```

As you may suspect, each of these resized graphs looks good. The overall graph looks good, and all the resized elements look good.

To recap, Stata provides numerous ways that you can size elements, giving you great control over

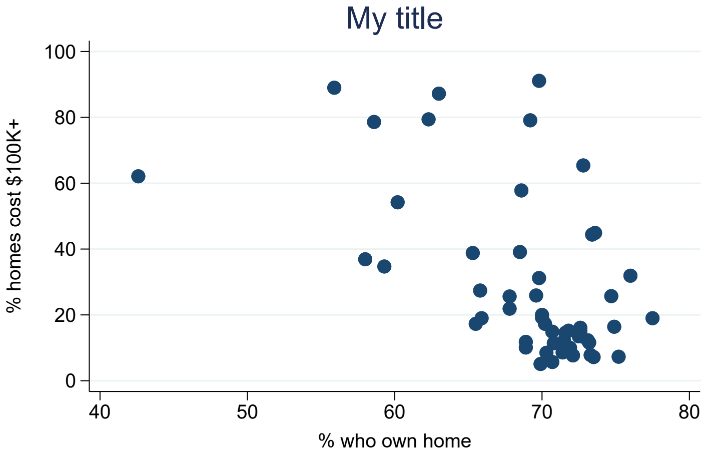
1. the sizing of titles, subtitles, and other text elements; see Options : Textboxes (section 8.11) and Styles : Textsize (section 10.12).
2. the sizing of axis titles and axis labels; see Options : Axis titles (section 8.4), Options : Axis labels (section 8.5), and Styles : Textsize (section 10.12).
3. the sizing of markers; see Options : Markers (section 8.1) and Styles : Markersize (section 10.9).
4. the sizing of marker labels; see Options : Marker labels (section 8.2) and Styles : Textsize (section 10.12).
5. the width of lines; see Styles : Linewidth (section 10.7).
6. the size of margins; see Styles : Margins (section 10.8).

Each of these elements can be sized using absolute units (that is, points, inches, and centimeters) or relative units (such as `size(large)`, `size(*2)`, or `size(5rs)`). The method you choose for sizing should match your purpose. Using absolute units (like points, inches, and centimeters) is useful when we know the exact size of the graph and the specifications of the exact sizing of the elements. In those cases, the `xsize()` and `ysize()` specifications will yield a graph that has the prespecified overall size, and each element will be sized exactly as specified. In some situations, like commercial art, such precise sizing to the 72nd of an inch may be both important and essential. But, if you will be resizing a graph via `xsize()` and `ysize()`, I think it may be easier to use relative units for specifying the size of the graphical elements.

9.4 Changing the look of graph regions

This section discusses the region options that we can control with the `plotregion()` and `graphregion()` options. These options allow us to control the plot region and graph region color, as well as the lines that border these regions. For more information, see `help region_options`.

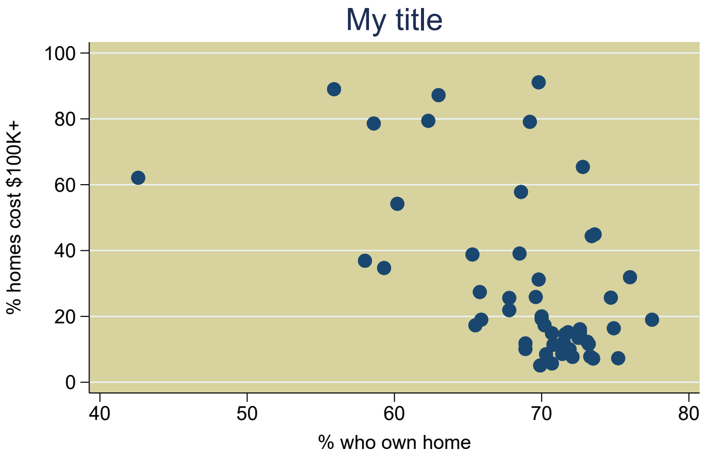
```
scatter propval100 ownhome, title("My title")
```



Consider this scatterplot. In general, Stata sees this graph as having two overall regions. The area inside the x and y axes where the data are plotted is called the *plot region*. In this graph, the plot region is white. The area surrounding the plot region, where the axes and titles are placed, is called the *graph region*. Here the graph region is shaded light blue. *Uses allstates.dta*

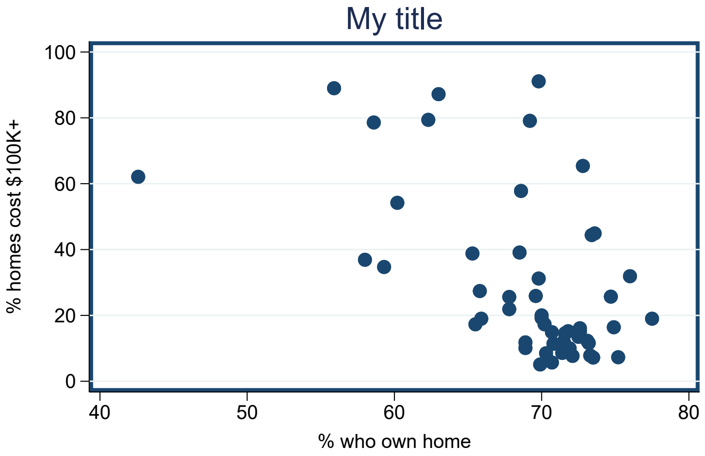
```
scatter propval100 ownhome, title("My title")
plotregion(color(stone))
```

Here we use `plotregion(color(stone))` to make the color of the plot region stone. The `color()` option controls the color of the plot region. *Uses allstates.dta*



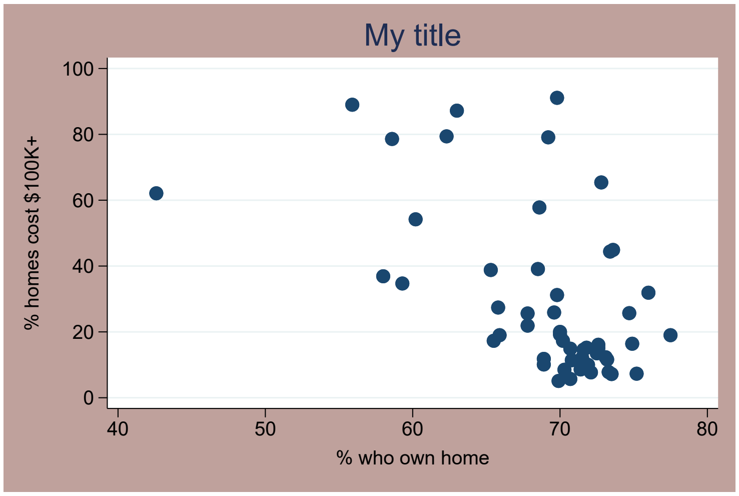
```
scatter propval100 ownhome, title("My title")
plotregion(lcolor(navy) lwidth(thick) )
```

In this graph, we put a thick, navy blue line around the plot region by using the `lcolor()` and `lwidth()` options. This puts a bit of a frame around the plot region. *Uses allstates.dta*

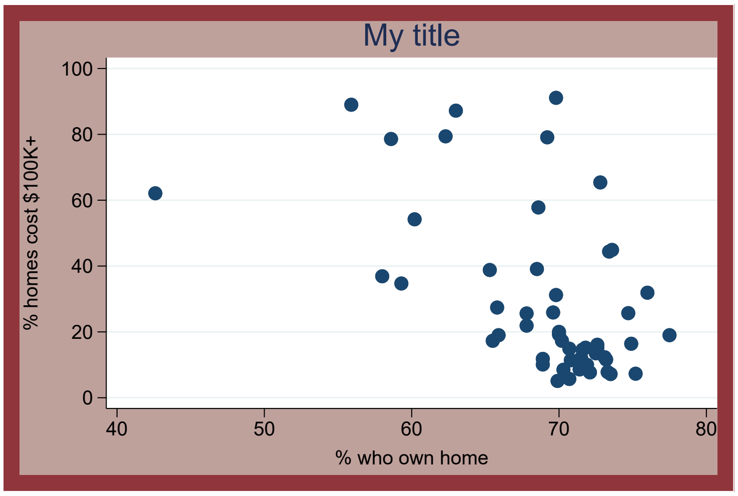


```
scatter propval100 ownhome, title("My title")
graphregion(color(erose))
```

Here we use the `graphregion(color(erose))` option to modify the color of the graph region to be erose, a light rose color. The graph region is the area outside the plot region where the titles and axes are displayed. *Uses allstates.dta*



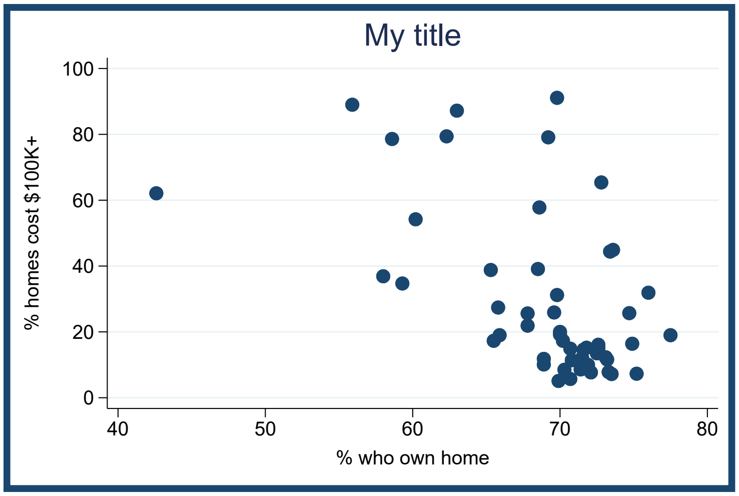
```
scatter propval100 ownhome, title("My title")
graphregion(ifcolor(erose) fcolor(maroon))
```



The graph region is actually composed of an inner part and an outer part. Here we use the `ifcolor(erose)` option to make the inner graph region light rose and the `fcolor(maroon)` option to make the outer graph region maroon. *Uses allstates.dta*

```
scatter propval100 ownhome, title("My title")
graphregion(lcolor(navy) lwidth(vthick))
```

We can put a somewhat different frame around the graph by altering the size and color of the line that surrounds the graph region. Using the `lcolor(navy) lwidth(vthick)` options gives this graph a very thick, navy blue border. *Uses allstates.dta*



This section omitted many options that you could use to control the plot region and graph region, including more control of the inner and outer regions and more control of the lines that surround these regions. Stata gives you more control than you generally need, so rather than covering these options here, I refer you to `help region_options`.

Chapter 10
Styles for changing the look of graphs

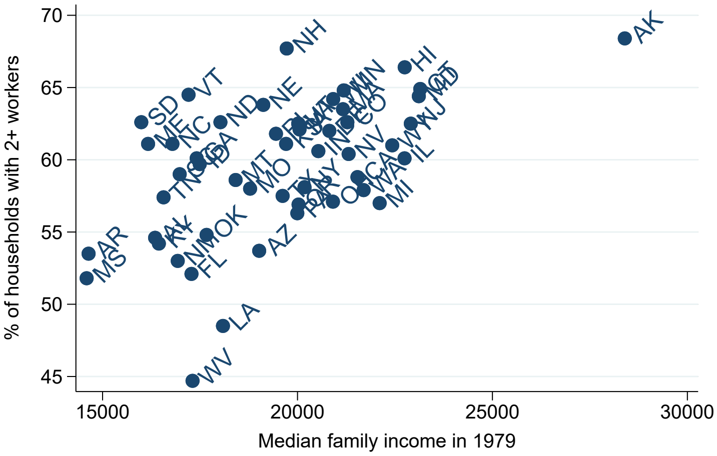
This chapter focuses on frequently used styles that arise in making graphs, such as *linepatternstyle*, *linewidthstyle*, or *markerstyle*. The chapter covers styles in alphabetical order, providing more details about the values we can choose. Each section refers to the corresponding section of the *Graphics Reference Manual*, which provides complete details on each style. We begin by using the `allstatesdc` file, which contains the `allstates` data with Washington, DC, omitted.

10.1 Angle

An *anglestyle* specifies the angle for displaying an item (or group of items) in the graph. Common examples include specifying the angle for marker labels with `mlabangle()` or the angle of the labels on the y axis with `ylabel(, angle())`. We can specify an *anglestyle* as a number of degrees of rotation (negative values are permitted, so for example, -90 can be used instead of 270). We can also use the keywords `horizontal` for 0 degrees, `vertical` for 90 degrees, `rhorizontal` for 180 degrees, and `rvertical` for 270 degrees. See `help anglestyle` for more information.

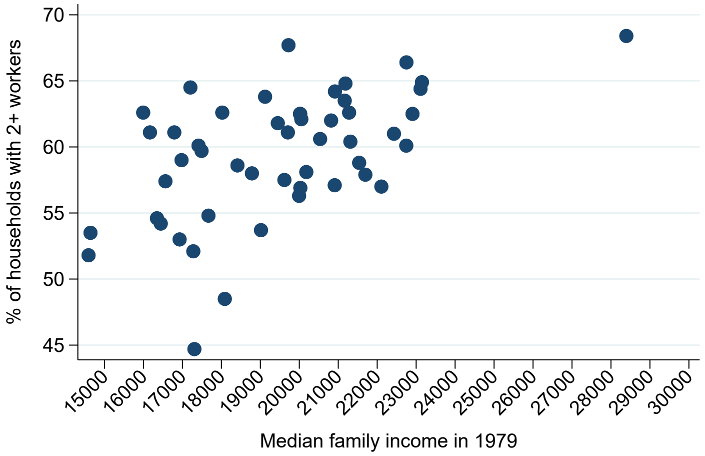
```
scatter workers2 faminc, mlabel(stateab) mlabangle(45)
```

Here we use the `mlabangle(45)` (marker label angle) to change the angle of the marker labels to 45 degrees. *Uses allstatesdc.dta*

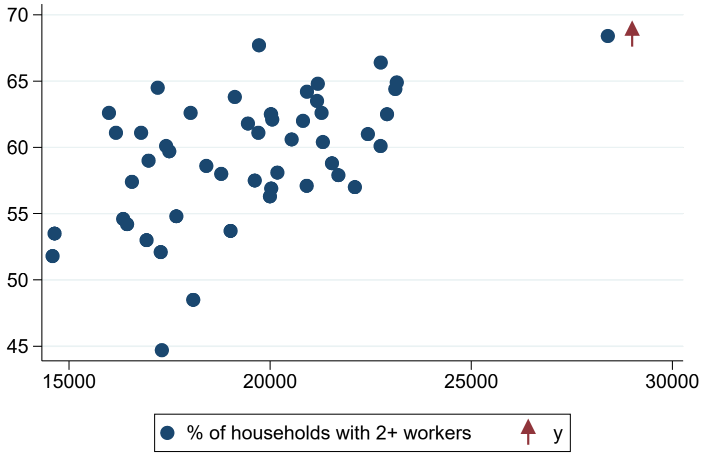


```
scatter workers2 faminc, xlabel(15000(1000)30000, angle(45))
```

In this example, we label the x axis from 15,000 to 30,000, incrementing by 1,000. When we have so many labels, we can use the `angle(45)` option to display the labels at a 45-degree angle. *Uses allstatesdc.dta*



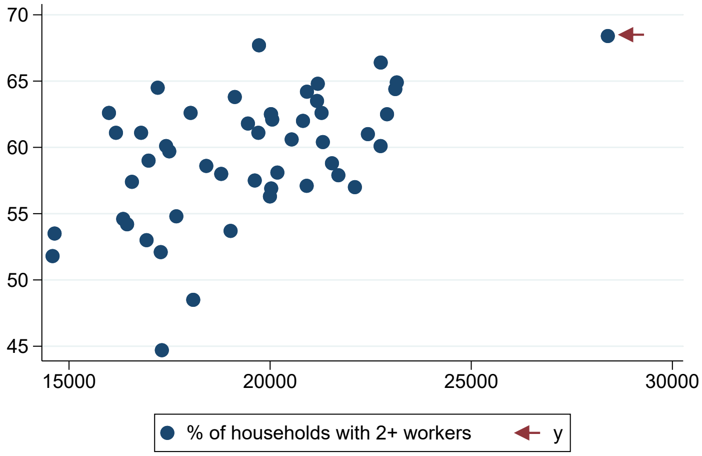
```
twoway (scatter workers2 faminc) (scatteri 68.5 29000,
msymbol(A) msize(huge))
```



In the prior example, one of the observations looks like a potential outlier. I would like to draw an arrow pointing at that observation to call attention to it. My first attempt uses the `scatteri` command to add an additional observation that uses a huge arrow as the marker symbol. I want the arrow to point to the left, but the default orientation for such an arrow is to point to the top of the graph. *Uses allstatesdc.dta*

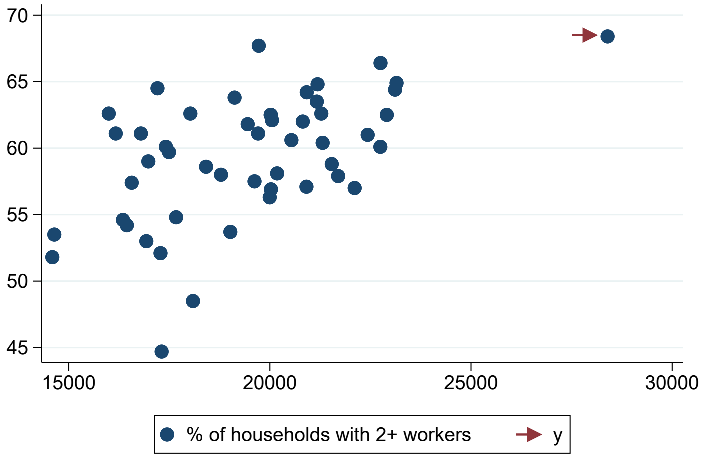
```
twoway (scatter workers2 faminc) (scatteri 68.5 29000,
msymbol(A) msize(huge) msangle(90))
```

In this example, I added the `msangle(90)` option to specify the marker symbol angle. This option rotates the marker symbol counterclockwise. Rotating the arrow 90 degrees, counterclockwise, makes the arrow point at the potential outlier. *Uses allstatesdc.dta*



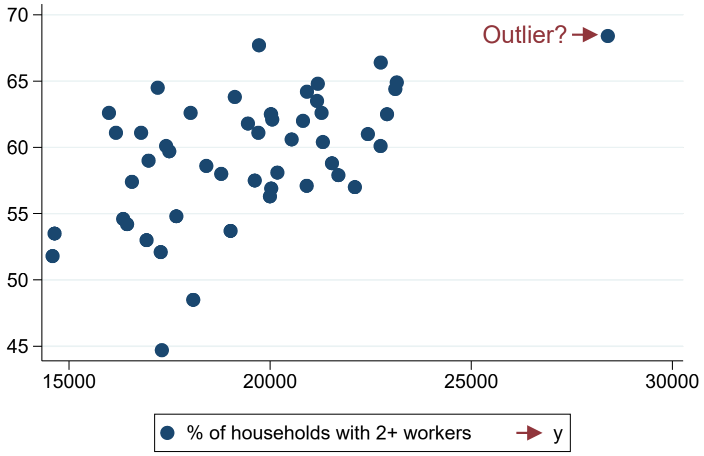
```
twoway (scatter workers2 faminc) (scatteri 68.5 27800,
msymbol(A) msize(huge) msangle(270))
```

This example is like the prior example; however, I have positioned the arrow to the left of the potential outlier. In this case, the `msangle(270)` option rotates the marker symbol 270 degrees counterclockwise, making the arrow point to the right and pointing at the potential outlier. *Uses allstatesdc.dta*



```
twoway (scatter workers2 faminc) (scatteri 68.5 27800 (9)
"Outlier?", msymbol(A) msize(huge) msangle(270))
```

I have extended the prior example, adding the label `Outlier?` positioned at the 9 o'clock position with respect to the arrow. This places the word `Outlier?` just to the left of the arrow. *Uses allstatesdc.dta*


70
65
60
55
50
45
15000
20000
25000
30000
Outlier?
● % of households with 2+ workers
y

10.2 Color

Stata gives you the ability to control the color of markers, lines, bars, boxes, titles, lines around boxes, fill colors of boxes, and practically any other object in a graph. Additionally, Stata allows you to control not just the color of objects but also the intensity (brightness) of the colors, as well as their opacity/transparency. In this section, I illustrate 1) how you can specify the color of objects, 2) the intensity of the color (that is, its brightness or darkness), and 3) the level of opacity/transparency used to display the object. I will begin by showing how you can control colors using named keywords that specify commonly used colors (see section 10.2.1). Next, I will illustrate how you can control the intensity of colors, showing how you can increase the brightness of colors (see section 10.2.2) and showing how you can decrease the brightness of colors (see section 10.2.2). The following section introduces the concept of opacity (the inverse of transparency) and shows how you can control the opacity/transparency of colors (see section 10.2.3). Then, I cover additional, more technical, ways that you can specify colors in Stata, including how to specify colors using RGB values, CMYK values, and HSV/HSL/HSB (see section 10.2.5). That section also illustrates how to incorporate intensity multipliers to modify the intensity and adjustments to opacity when specifying colors using these methods.

My examples will mainly use the `graph bar` command to illustrate how to specify colors and show the resulting colors that arise using different color specifications. I chose `graph bar` because 1) the colors are easy to see on a bar (versus a point on a scatterplot), 2) it is easy to show multiple colors in one graph, 3) bars are, by default, immediately adjacent to one another, facilitating comparison of colors in adjacent bars, and 4) bars can be made to overlap, illustrating the impact of opacity/transparency when there is overlap of objects. Nevertheless, the methods I illustrate for specifying color/intensity/opacity apply to all graph types and apply to any graphical element where you can control the color. Finally, in this book I never abbreviate command names or option names for maximum clarity. However, in some of the examples below, I have frequently abbreviated the `color()` option to `col()` solely because of space considerations, trying to make the long commands fit on the printed page. See `help colorstyle` for more information.

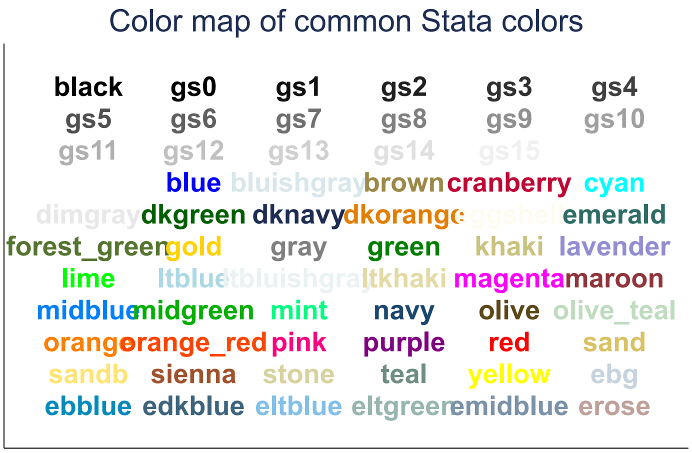
10.2.1 Named colors

You can specify a color using a keyword that names a color. You can get a list of the keywords for naming colors with the `graph query colorstyle` command. I have executed that command below, and the output shows the named colors available when I executed this command. It is possible that when you run this command, there may be additional keywords available for specifying colors.

```
 . graph query colorstyle

colorstyle may be

    black          emerald        gs5            none
    blue           emidblue       gs6            olive
    bluishgray     erose          gs7            olive_teal
    bluishgray8    forest_green   gs8            orange
    brown          gold           gs9            orange_red
    chocolate      gray           khaki          pink
    cranberry      green          lavender       purple
    cyan           gs0            lime           red
    dimgray        gs1            ltblue         sand
    dkgreen        gs10           ltbluishgray   sandb
    dknavy         gs11           ltbluishgray8  sienna
    dkorange       gs12           ltkhaki        stone
    ebblue         gs13           magenta        sunflowerlime
    ebg            gs14           maroon         teal
    edkbg          gs15           midblue        white
    edkblue        gs16           midgreen       yellow
    eggshell       gs2            mint
    eltblue        gs3            navy
    eltgreen       gs4            navy8

    For information on colorstyle and how to use it, see help colorstyle.
```

Seeing this list of keywords for choosing colors is useful, but it is like going to a foreign restaurant and seeing a menu with a bunch of names of dishes. In such cases, I like the menus that also show the picture of the prepared dish. It gives me an idea of what to expect. In that spirit, I have written a command called `vgcolormap` that shows the keywords for commonly used colors displayed using the specified color. For example, in the graph below, you can see not only that `red` is a keyword but also that keyword is displayed in red, showing you how it will appear on your screen. (Note that I intentionally made the color names so large that some of the long names overlap, because I wanted the type to be large enough that colors are extremely visible.)

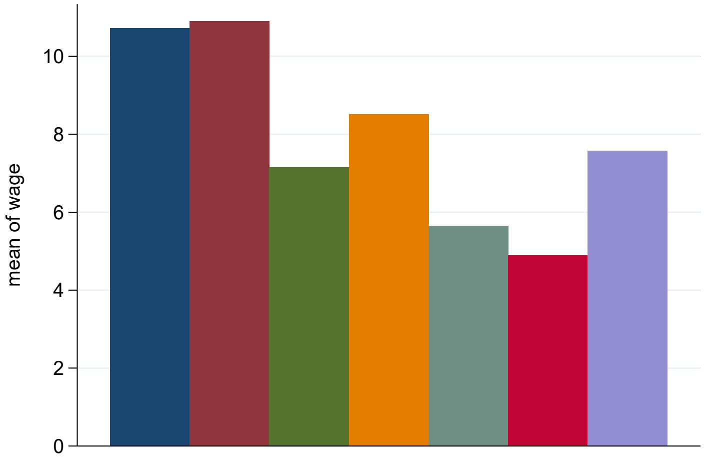
```
vgcolormap, quietly
```



The author-written `vgcolormap` command shows the different standard colors available in Stata. We simply issue the command `vgcolormap`, and it creates a scatterplot that shows the colors we can choose and their names. To see the entire list of colors and for more information about colors, see `help colorstyle`. Also, see Introduction : Online supplements (section 1.1) to learn how to get `vgcolormap`. *Uses nlsw.dta*

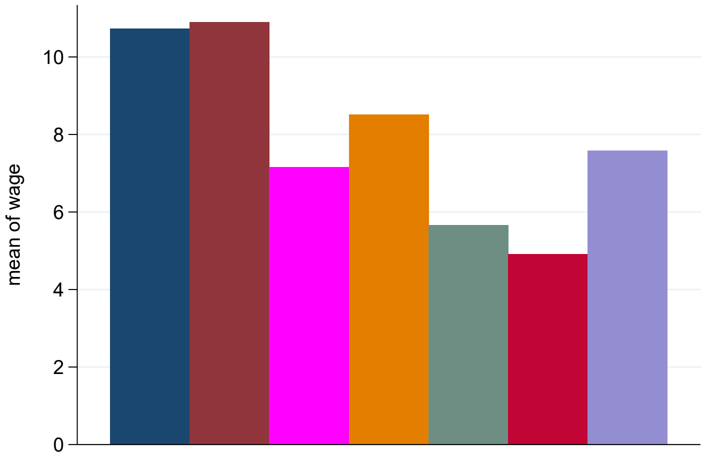
```
graph bar wage, over(occ7) asyvars legend(off)
```

I will use the `graph bar` command as my main tool for showing how to specify colors and displaying what those colors look like. Note how there are seven bars, each shown in a different color, corresponding to the seven levels of `occ7`. In other examples, I will use `occ5` when I want to show five colors, and I will use `occ10` when I want to show 10 colors at one time. The legend is not meaningful for these examples, so I will omit it by using the `legend(off)` option throughout this section. *Uses nlsw.dta*



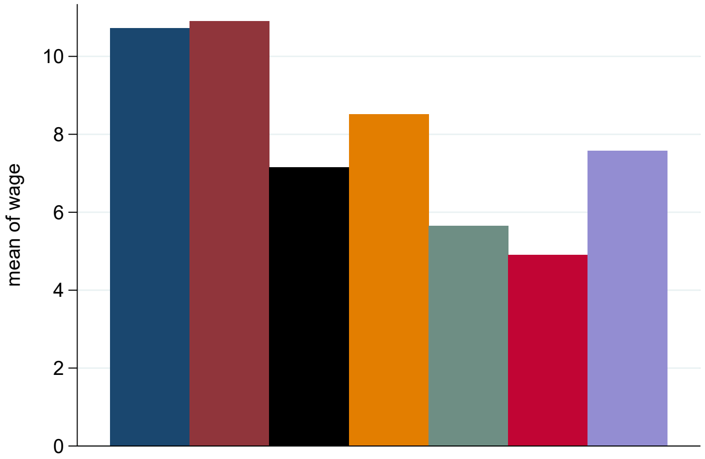
```
graph bar wage, over(occ7) asyvars legend(off) bar(3,
color(magenta))
```

Say that I am mad at the people in Sales, and to show this I want to make the color of their bar a peculiar color. In this example, I use the `bar(3, color(magenta))` option to make the third bar color `magenta`. *Uses nlsw.dta*



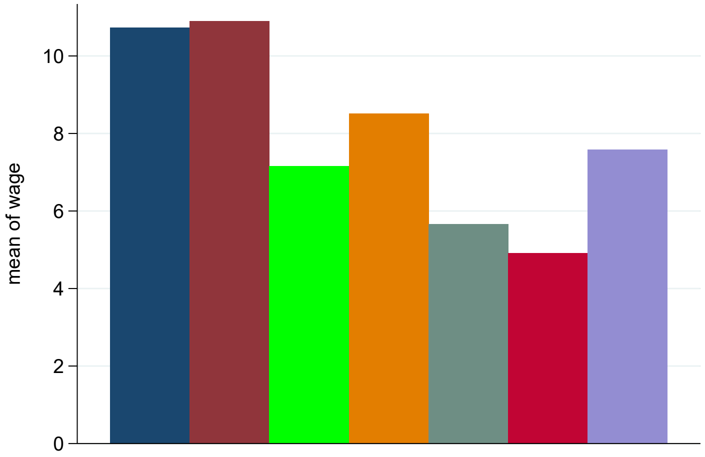
```
graph bar wage, over(occ7) asyvars legend(off) bar(3,
color(limme))
```

I decide that I want to use an even more peculiar color for sales, so I decide to make the color for that bar `lime`. At first, I am confused by this graph because the bar for Sales is shown in black. I see that I got an error from this command, saying that `limme` was not found. Because I misspelled the color, a default color (black) was used. *Uses nlsw.dta*



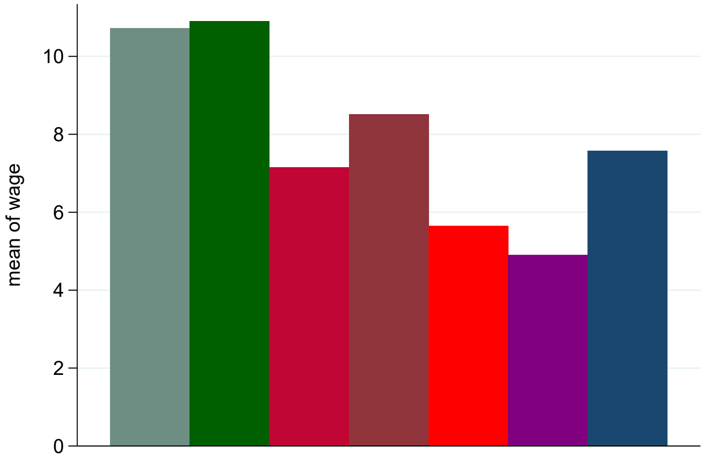
```
graph bar wage, over(occ7) asyvars legend(off) bar(3,
color(lime))
```

Now, I fix the color to, say, `lime`, and the bar for Sales is shown in lime. *Uses nlsw.dta*

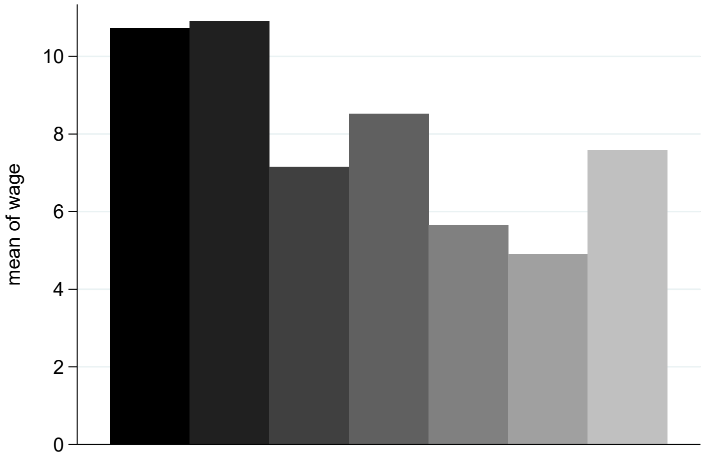


```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(teal)) bar(2, color(dkgreen)) bar(3, color(cranberry))
bar(4, color(maroon)) bar(5, color(red)) bar(6,
color(purple)) bar(7, color(navy))
```

So far, my examples have illustrated only changing the color of a single bar. With the `graph bar` command, I can use the `color()` suboption to specify the color for each of the bars. In this example, I have specified the `color()` suboption for each of the bars. *Uses nlsw.dta*



```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(gs0)) bar(2, color(gs2)) bar(3, color(gs4)) bar(4,
color(gs6)) bar(5, color(gs8)) bar(6, color(gs10)) bar(7,
color(gs12))
```



We can create bars using shades of black. The graph created by `vgcolormap` from section 10.2.1 shows 16 keyword names for 16 shades of gray, ranging from `gs0` (completely black) to `gs16` (completely white). This example uses `color(gs0)` for the first bar and `color(gs12)` for the last bar and the colors `gs2`, `gs4`, `gs6`, `gs8`, and `gs10` for the bars in between. *Uses nlsw.dta*

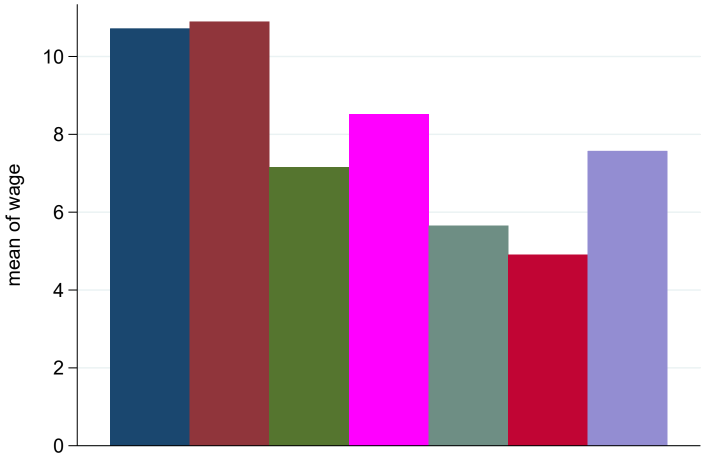
10.2.2 Color intensity

In the previous section, I showed how you can choose among different colors. For example, in one of the graphs above, I used the `color(navy)` option to make one of the bars display as navy blue. In addition to specifying the bar color as `navy`, I can also adjust the intensity of that color.

To do so, I could follow keyword `navy` with an `*` and then include a multiplier value to adjust the intensity. A multiplier of 1 leaves the value unchanged, while a value greater than 1 makes the color darker, and a value less than 1 makes the value brighter. A value of 0 is the brightest value (white or nearly white) and a value of 255 is the darkest value (black or nearly black).

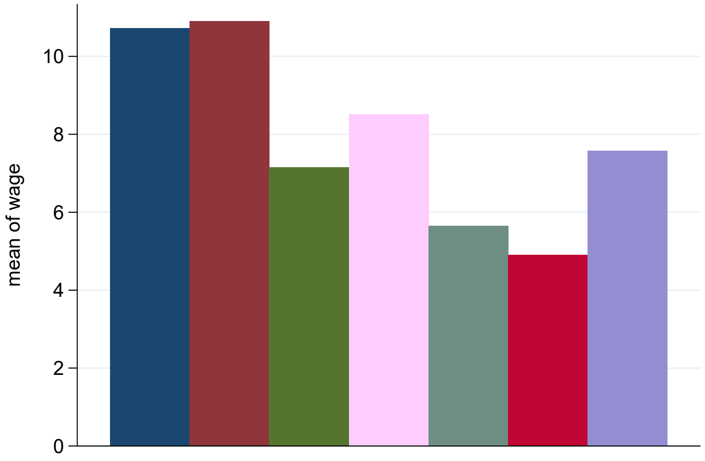
```
graph bar wage, over(occ7) asyvars legend(off) bar(4,
color(magenta))
```

Consider this example bar graph. The variable `occ7` has five levels. Because we used the `asyvars` option, each bar is shown using a different color. In this example, the `bar(4, color(magenta))` option specifies that the fourth bar will be displayed in `magenta`. *Uses nlsw.dta*



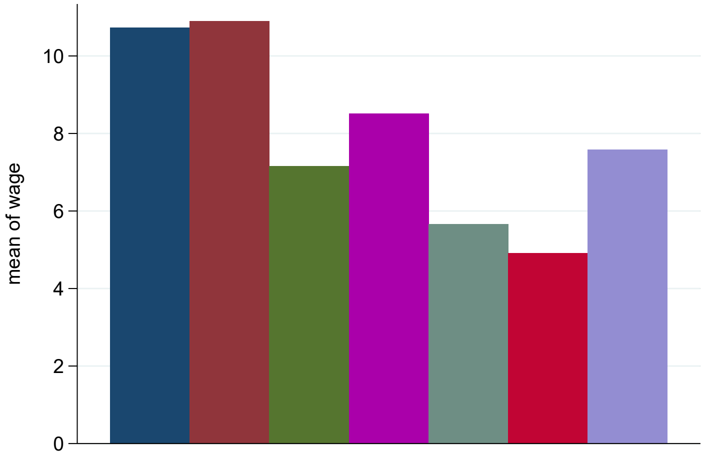
```
graph bar wage, over(occ7) asyvars legend(off) bar(4,
color(magenta*.2))
```

By specifying `color(magenta*.2)`, we display the fourth bar in magenta, but now it is brighter (lighter) than normal (because an intensity multiplier of less than 1 was specified). Specifying values greater than 1 reduces brightness, as illustrated in the next graph. *Uses nlsw.dta*



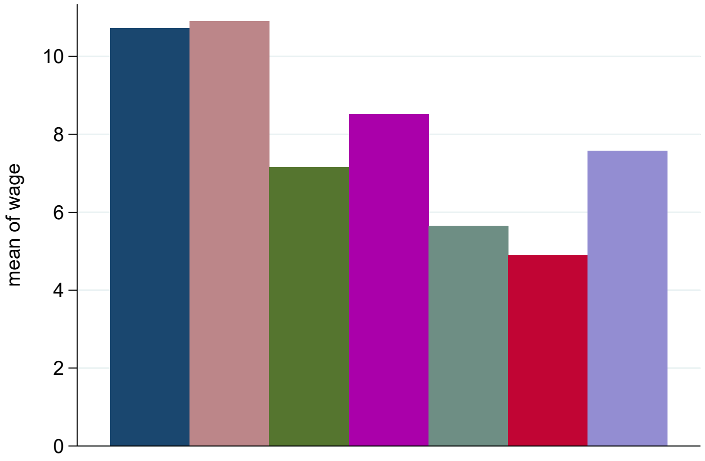
```
graph bar wage, over(occ7) asyvars legend(off) bar(4,
color(magenta*1.5))
```

Now, when we specify `color(magenta*1.5)`, the intensity multiplier for the fourth bar is now greater than 1, so now the fourth bar is displayed in magenta but with less brightness than normal (in other words, darker than normal). *Uses nlsw.dta*

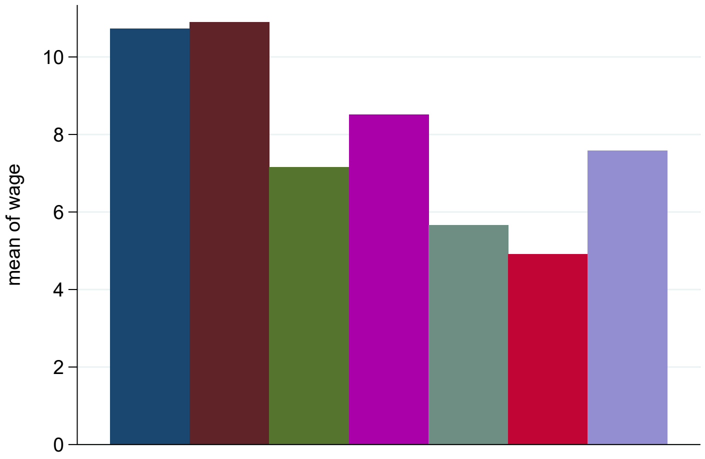


```
graph bar wage, over(occ7) asyvars legend(off) bar(4,
color(magenta*1.5)) bar(2, color(*.6))
```

Suppose we like the color of the second bar, but we just want it to be displayed brighter than normal. To do that, we want to apply an intensity multiplier that is less than 1. In this example, I specify the option `bar(2, color(*.6))` to display the second bar using its default color but displaying it brighter than normal. *Uses nlsw.dta*



```
graph bar wage, over(occ7) asyvars legend(off) bar(4,
color(magenta*1.5)) bar(2, color(*1.5))
```



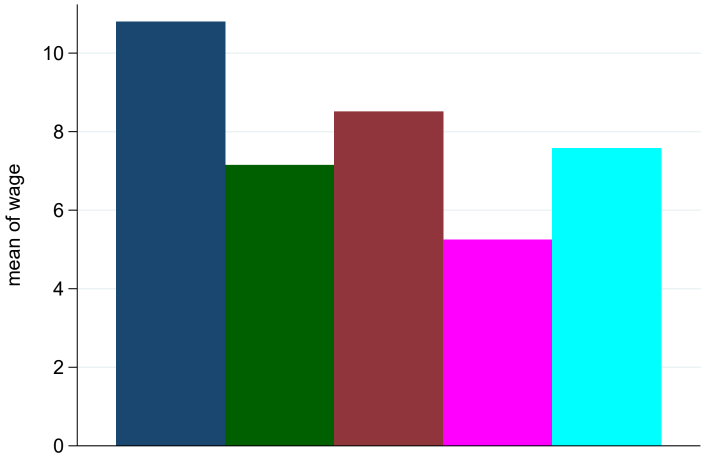
Suppose we change our mind and decide to show the second bar using its default color but instead showing it darker than normal. By specifying `bar(2, color(*1.5))`, we show the second bar with its default color but with an intensity multiplier of `1.5`, making the color less bright (darker). *Uses nlsw.dta*

The following examples will illustrate the impact of increasing brightness (using an intensity multiplier less than 1) followed by examples of decreasing brightness (using an intensity multiplier greater than 1).

Color intensity: Increasing brightness

These examples will use the `graph bar` command to illustrate how to increase color intensity, increasing the brightness of colors. The first example shows a bar chart with five colors, `navy`, `dkgreen`, `maroon`, `magenta`, and `cyan`. Then, for each of these colors, I will make a bar chart with just one color (for example, `navy`) using different levels of intensity for each of the bars. I chose these colors on purpose because they vary in their natural brightness. This will allow us to see how increasing intensity (brightness) impacts different kinds of colors.

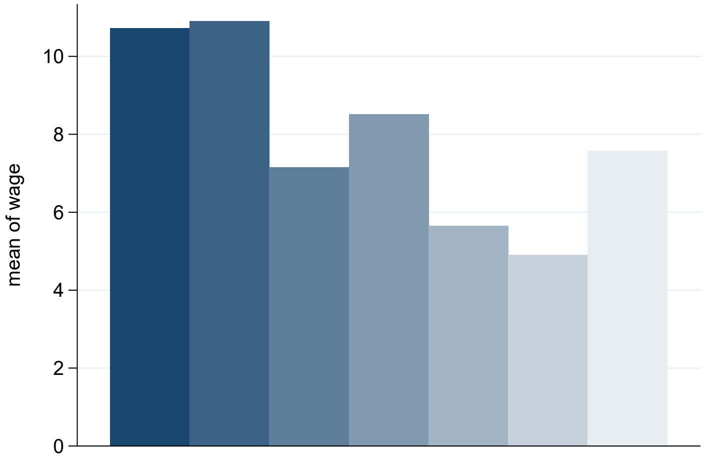
```
graph bar wage, over(occ5) asyvars legend(off) bar(1,
color(navy)) bar(2, color(dkgreen)) bar(3, color(maroon))
bar(4, color(magenta)) bar(5, color(cyan))
```



Consider this bar chart. It shows the average wages across five occupations. By including the `asyvars` option, we show each occupation using a different color. I have used the `color()` suboption to make the color of the first bar `navy`, the second `dkgreen`, the third `maroon`, the fourth `magenta`, and the fifth `cyan`. *Uses nlsw.dta*

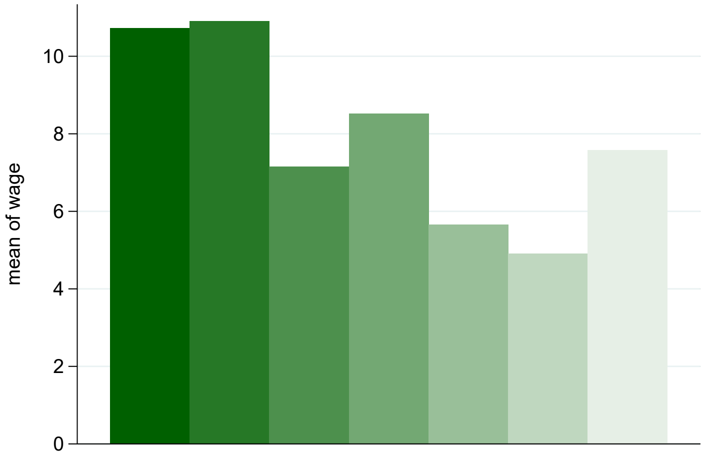
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(navy*1.0)) bar(2, color(navy*0.85)) bar(3,
color(navy*0.7)) bar(4, color(navy*0.55)) bar(5,
color(navy*0.4)) bar(6, color(navy*0.25)) bar(7,
color(navy*0.1))
```

This bar graph shows seven bars. Each bar is shown using the color `navy`, but using seven different intensity multipliers. The multipliers for the seven bars are `*1.0`, `*0.85`, `*0.7`, `*0.55`, `*0.4`, `*0.25`, and `*0.1`. You can see how the brightness increases as the multiplier decreases. *Uses nlsw.dta*



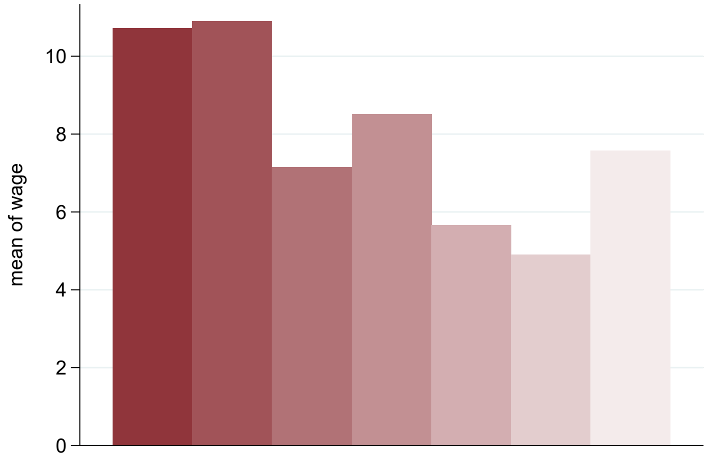
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(dkgreen*1.0)) bar(2, color(dkgreen*0.85)) bar(3,
color(dkgreen*0.7)) bar(4, color(dkgreen*0.55)) bar(5,
color(dkgreen*0.4)) bar(6, color(dkgreen*0.25)) bar(7,
color(dkgreen*0.1))
```

This graph is like the previous graph but displays seven bars using the color dark green (`dkgreen`). The first bar uses the normal intensity, and the intensity multipliers for the remaining bars are `*0.85`, `*0.7`, `*0.55`, `*0.4`, `*0.25`, and `*0.1`. You can see how the brightness increases as the multiplier decreases. Note the different intensities of `dkgreen` for each bar. *Uses nlsw.dta*



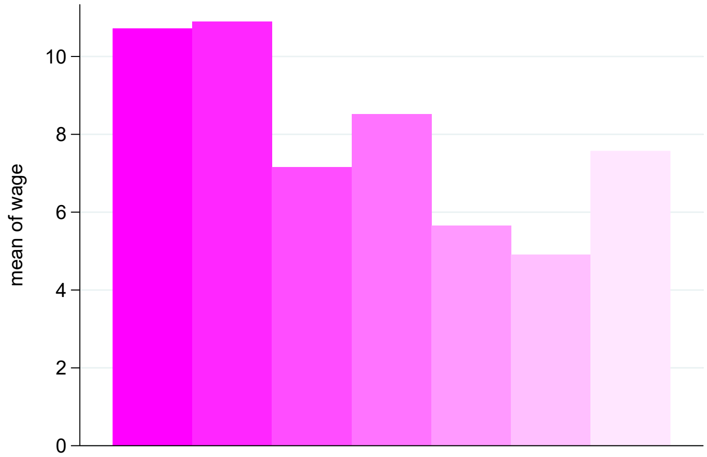
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(maroon*1.0)) bar(2, color(maroon*0.85)) bar(3,
color(maroon*0.7)) bar(4, color(maroon*0.55)) bar(5,
color(maroon*0.4)) bar(6, color(maroon*0.25)) bar(7,
color(maroon*0.1))
```

This example shows seven bars, each using color `maroon`, showing how the bars are displayed when using intensities of `*1.0`, `*0.85`, `*0.7`, `*0.55`, `*0.4`, `*0.25`, and `*0.1`. *Uses nlsw.dta*



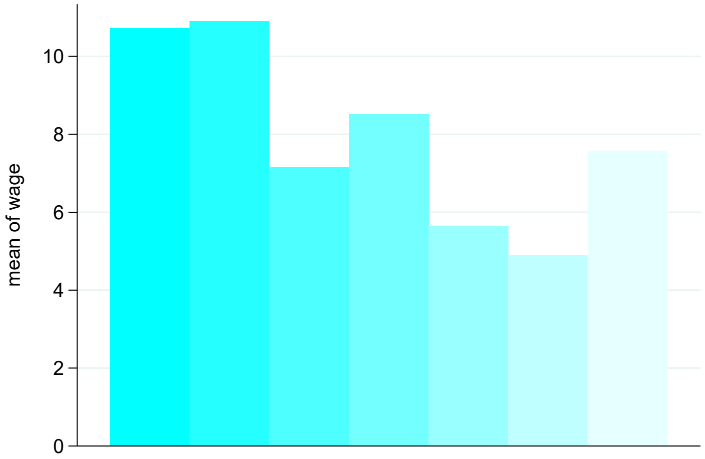
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(magenta*1.0)) bar(2, color(magenta*0.85)) bar(3,
color(magenta*0.7)) bar(4, color(magenta*0.55)) bar(5,
color(magenta*0.4)) bar(6, color(magenta*0.25)) bar(7,
color(magenta*0.1))
```

This example is just like the prior example, but now this shows how the color `magenta` looks at these seven different intensities. Note how the first and second bars look identical when displayed in `magenta`, while the first and second bars were rather different when displayed using `navy`, `dkgreen`, or `maroon`. *Uses nlsw.dta*



```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(cyan*1.0)) bar(2, color(cyan*0.85)) bar(3,
color(cyan*0.7)) bar(4, color(cyan*0.55)) bar(5,
color(cyan*0.4)) bar(6, color(cyan*0.25)) bar(7,
color(cyan*0.1))
```

Now, consider the look of the bars using these seven intensities when using a bright color, `cyan`. The first three bars are nearly identical in color, and the fourth is not much different from the third. *Uses nlsw.dta*



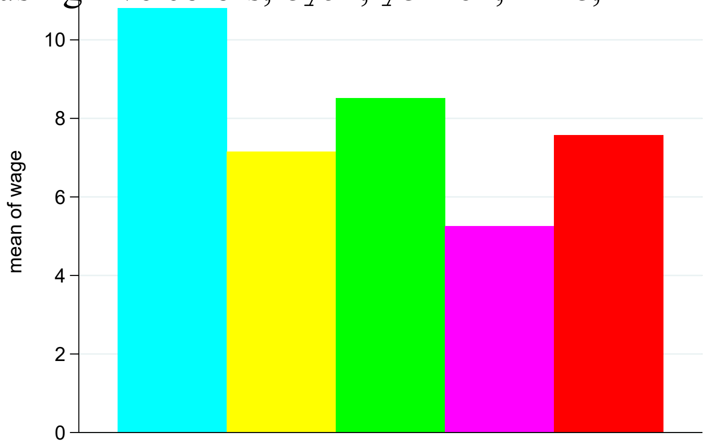
As we saw in the prior examples, increased brightness has a very perceptible impact on dark colors like `maroon`, `navy`, and `dkgreen`. In examples using those colors, we could clearly see the differences in bars shown using intensities of 1.0, 0.85, 0.7, 0.55, 0.4, 0.25, and 0.1. By contrast, it was hard to see the impact of increasing the brightness for a color that is already rather bright (like `cyan`).

Color intensity: Decreasing brightness

These examples illustrate how to alter intensity to decrease the brightness of colors. Or, framed differently, we will use intensity multipliers greater than 1 to increase the darkness of the colors. The format of these examples will be very similar to the examples where I illustrated increasing brightness, except that I will use the colors `cyan`, `yellow`, `lime`, `magenta`, and `red`. I will first make a bar chart with these five colors, and then I will make a separate bar chart using each color where I vary the intensity of the color (making the colors darker). As before, I chose these colors because they are mostly bright colors, but they also vary in their natural level of brightness. This will allow us to see how increasing the intensity of different kinds of colors impact their appearance.

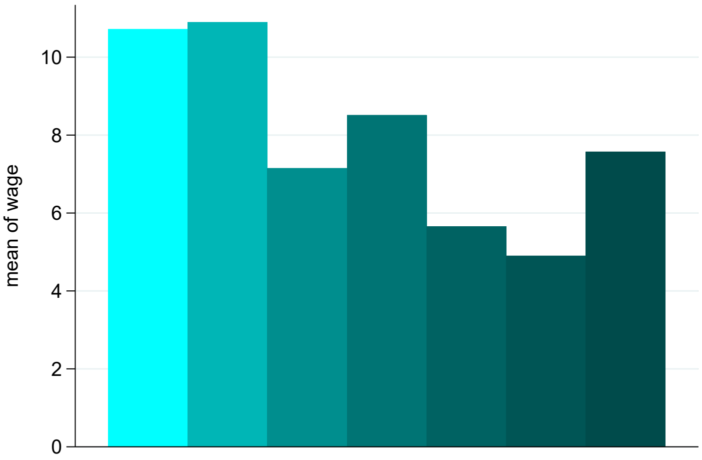
```
graph bar wage, over(occ5) asyvars legend(off) bar(1,
color(cyan)) bar(2, col(yellow)) bar(3, col(lime)) bar(4,
col(magenta)) bar(5, col(red))
```

Consider this bar graph shown using five colors, `cyan`, `yellow`, `lime`, `magenta`, and `red`. In the following examples, I will explore how these colors look when we use intensity multipliers greater than 1 that decrease the brightness of the color. *Uses nlsw.dta*



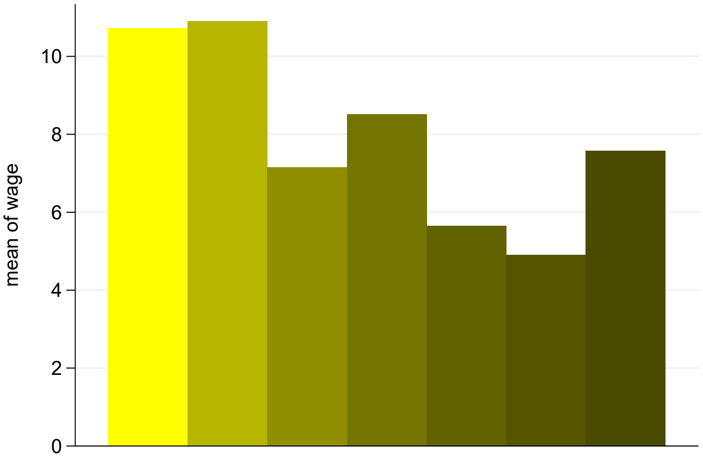
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
color(cyan*1.0)) bar(2, col(cyan*1.4)) bar(3, col(cyan*1.8))
bar(4, col(cyan*2.2)) bar(5, col(cyan*2.6)) bar(6,
col(cyan*3.0)) bar(7, col(cyan*3.4))
```

This example shows bars displayed using the color `cyan` using intensity multipliers ranging from 1.0 to 3.4 in increments of 0.4. I am intrigued by the appearance of these bars using these different intensities of `cyan`. *Uses nlsw.dta*



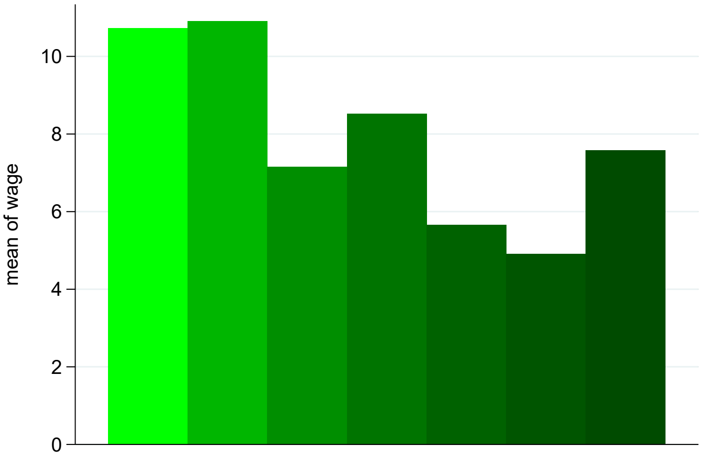
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
col(yellow*1.0)) bar(2, col(yellow*1.4)) bar(3,
col(yellow*1.8)) bar(4, col(yellow*2.2)) bar(5,
col(yellow*2.6)) bar(6, col(yellow*3.0)) bar(7,
col(yellow*3.4))
```

In this graph, all seven bars are shown in `yellow`. But, like the prior example, the seven bars are displayed using intensity multipliers ranging from 1.0 to 3.4 in increments of 0.4. As I look at these colors, I am struck by how different these shades of `yellow` appear and how much I feel that my array of color choices is expanded by exploring these shades of yellow. *Uses nlsw.dta*



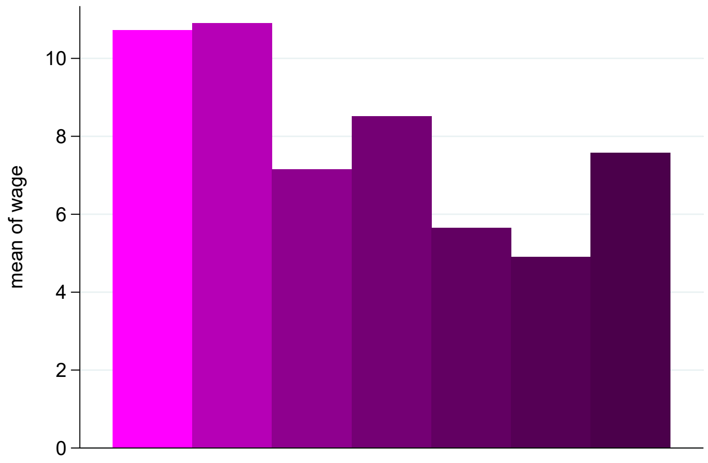
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
col(lime*1.0)) bar(2, col(lime*1.4)) bar(3, col(lime*1.8))
bar(4, col(lime*2.2)) bar(5, col(lime*2.6)) bar(6,
col(lime*3.0)) bar(7, col(lime*3.4))
```

This graph shows the color `lime` using the intensity multipliers ranging from 1.0 to 3.4 in increments of 0.4. This graph is almost exactly what I pictured in my mind before looking at the graph, showing shades of green ranging from a bright green to a dark green. *Uses nlsw.dta*



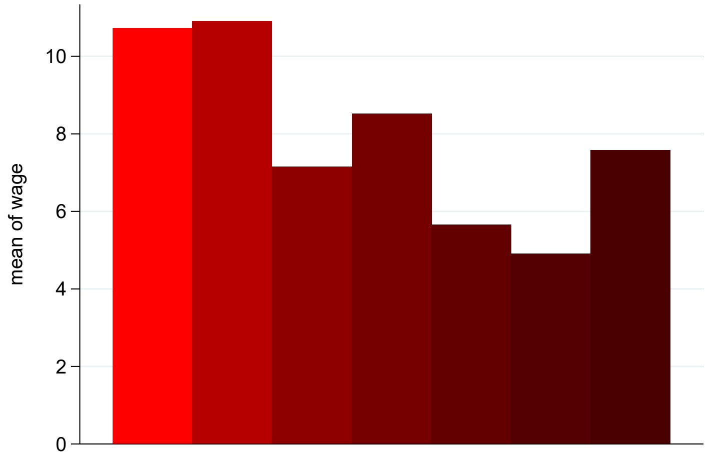
```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
col(magenta*1.0)) bar(2, col(magenta*1.4)) bar(3,
col(magenta*1.8)) bar(4, col(magenta*2.2)) bar(5,
col(magenta*2.6)) bar(6, col(magenta*3.0)) bar(7,
col(magenta*3.4))
```

As I mentioned before, I find `magenta` to be a very whimsical color, so I wanted to explore the appearance of `magenta` using intensity multipliers ranging from 1.0 to 3.4 in increments of 0.4. There are some very fun shades of magenta among these seven bars. *Uses nlsw.dta*



```
graph bar wage, over(occ7) asyvars legend(off) bar(1,
col(red*1.0)) bar(2, col(red*1.4)) bar(3, col(red*1.8))
bar(4, col(red*2.2)) bar(5, col(red*2.6)) bar(6,
col(red*3.0)) bar(7, col(red*3.4))
```

This bar graph shows the color `red` using seven different intensity multipliers ranging from 1.0 to 3.4 in increments of 0.4. There are some intriguing shades of red among these seven bars, although the differences among the last three bars are hard to see. *Uses nlsw.dta*

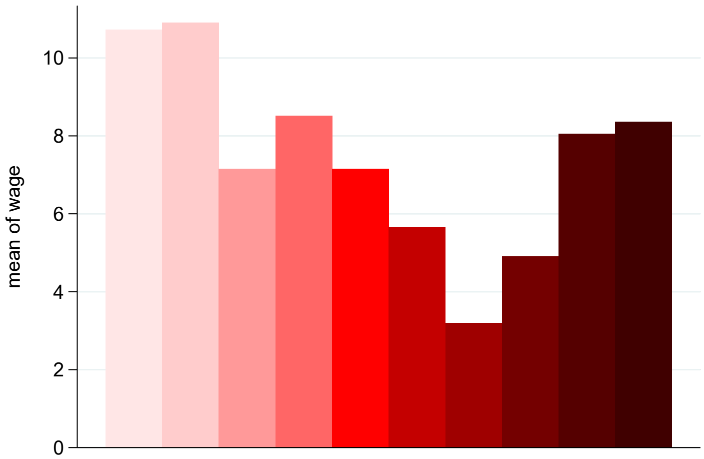


Color intensity: The spectrum of brightness

In the prior two sections, I showed examples of increasing brightness, focusing dark colors (like `navy`); I also showed examples of decreasing brightness, focusing on bright colors (like `yellow`). Now, I want to illustrate the impact of increasing and decreasing intensity (brightness). These examples will use the `graph bar` command to show 10 bars using the following intensity multipliers: `*0.1`, `*0.2`, `*0.4`, `*0.6`, `*1.0`, `*1.3`, `*1.6`, `*2.2`, `*3`, and `*4`. Each example will show one color across these 10 intensities. I hope that these examples serve two purposes. First, I hope they help to illustrate the rich variety of colors you can access by varying color intensity. Second, I hope you might use this strategy on your own to explore the impact of different intensities when using other colors.

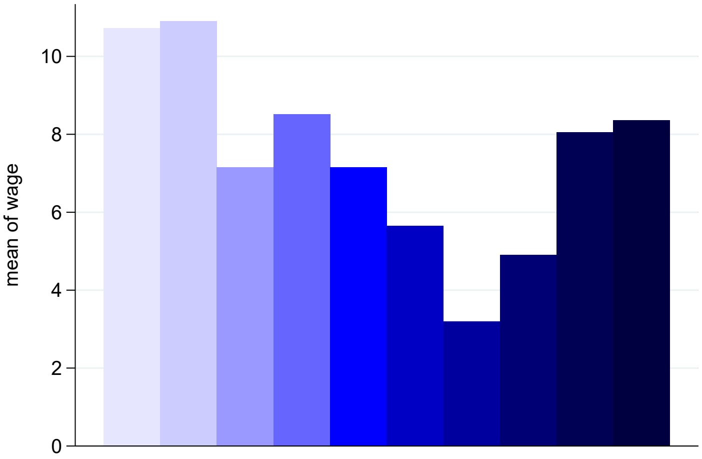
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(red*0.1)) bar(2, col(red*0.2)) bar(3, col(red*0.4))
bar(4, col(red*0.6)) bar(5, col(red*1.0)) bar(6,
col(red*1.3)) bar(7, col(red*1.6)) bar(8, col(red*2.2))
bar(9, col(red*3.0)) bar(10, col(red*4.0))
```

This shows the color red when using 10 different intensity multipliers: `*0.1`, `*0.2`, `*0.4`, `*0.6`, `*1.0`, `*1.3`, `*1.6`, `*2.2`, `*3`, and `*4`. Within these colors, I see many different shades, including shades that make me think of a very light pink, a tempting rosé wine, a fire engine, and a red brick. *Uses nlsw.dta*



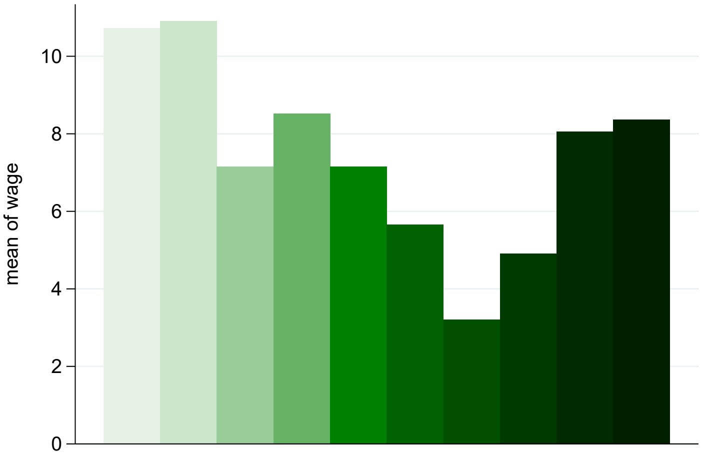
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(blue*0.1)) bar(2, col(blue*0.2)) bar(3, col(blue*0.4))
bar(4, col(blue*0.6)) bar(5, col(blue*1.0)) bar(6,
col(blue*1.3)) bar(7, col(blue*1.6)) bar(8, col(blue*2.2))
bar(9, col(blue*3.0)) bar(10, col(blue*4.0))
```

This example illustrates the color `blue` using the 10 intensity multipliers I used in the prior graph. *Uses nlsw.dta*



```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(green*0.1)) bar(2, col(green*0.2)) bar(3, col(green*0.4))
bar(4, col(green*0.6)) bar(5, col(green*1.0)) bar(6,
col(green*1.3)) bar(7, col(green*1.6)) bar(8, col(green*2.2))
bar(9, col(green*3.0)) bar(10, col(green*4.0))
```

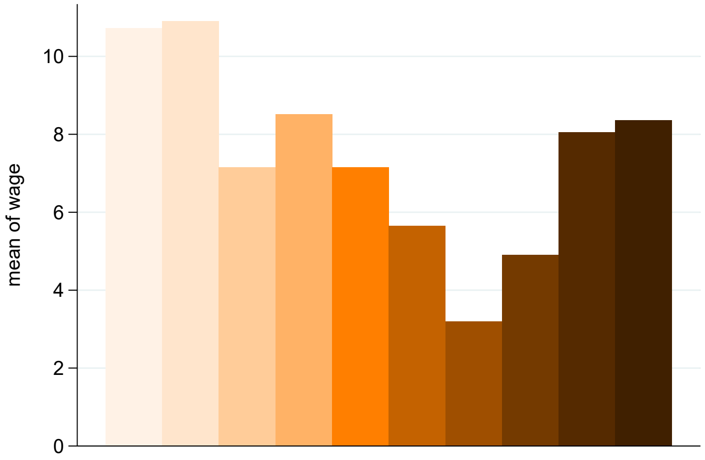
This graph shows the color `green` using the same 10 intensity multipliers from the prior example. *Uses nlsw.dta*



```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(orange*0.1)) bar(2, col(orange*0.2)) bar(3,
col(orange*0.4)) bar(4, col(orange*0.6)) bar(5,
col(orange*1.0)) bar(6, col(orange*1.3)) bar(7,
col(orange*1.6)) bar(8, col(orange*2.2)) bar(9,
col(orange*3.0)) bar(10, col(orange*4.0))
```

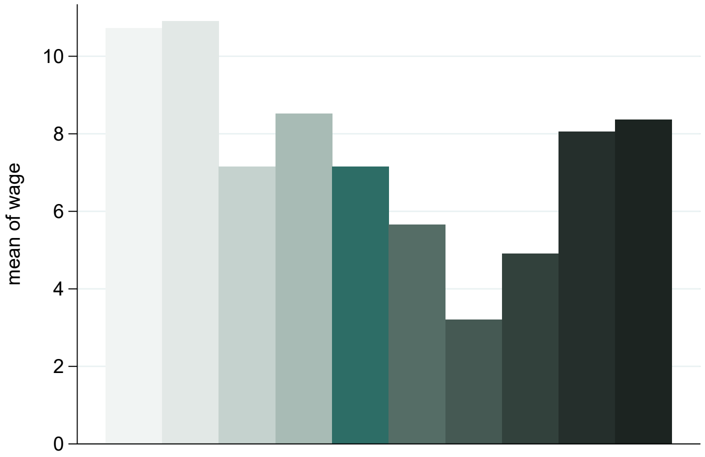
There are so many interesting colors among these 10 bars showing the color `orange`. The brighter versions remind me of peaches, and the darker colors

remind me of cinnamon.
Uses nlsw.dta



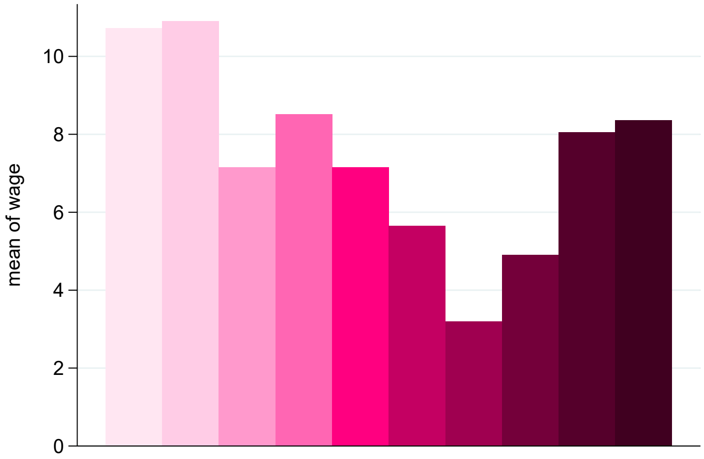
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(teal*0.1)) bar(2, col(teal*0.2)) bar(3, col(teal*0.4))
bar(4, col(teal*0.6)) bar(5, col(emerald*1.0)) bar(6,
col(teal*1.3)) bar(7, col(teal*1.6)) bar(8, col(teal*2.2))
bar(9, col(teal*3.0)) bar(10, col(teal*4.0))
```

This example illustrates the color `teal` using 10 different intensity multipliers. *Uses nlsw.dta*



```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(pink*0.1)) bar(2, col(pink*0.2)) bar(3, col(pink*0.4))
bar(4, col(pink*0.6)) bar(5, col(pink*1.0)) bar(6,
col(pink*1.3)) bar(7, col(pink*1.6)) bar(8, col(pink*2.2))
bar(9, col(pink*3.0)) bar(10, col(pink*4.0))
```

And this shows the color `pink` at 10 intensity levels. *Uses nlsw.dta*

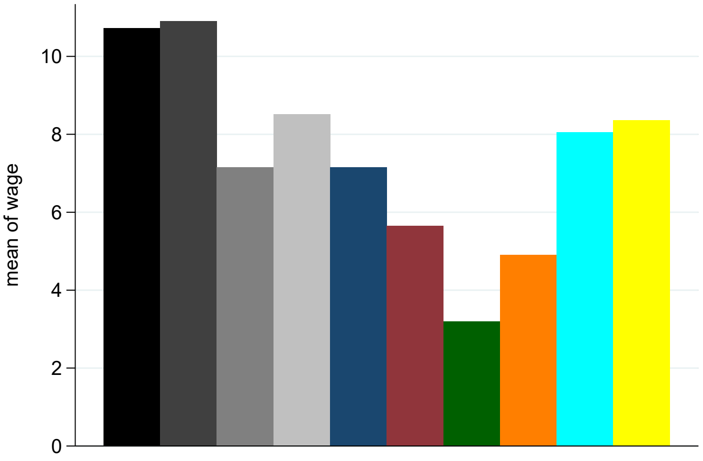


10.2.3 Color opacity (transparency)

In this section, I will show how you can adjust the *opacity* of colors. Opacity controls the percentage of the background we can see after a color has been applied. For some people, like me, I find it easier to think of this in terms of transparency. Opacity is the inverse of transparency. A color shown with 100% opacity covers the background completely—its transparency is 0%. A color shown with 0% opacity is displayed with 100% transparency. A color shown with 80% opacity (that is, 20% transparency) lets a little bit of the background show through, while a color shown with 20% opacity (that is, 80% transparency) lets lots of the background show through. In this section, I will use bar graphs to illustrate the impact of varying opacity (transparency).

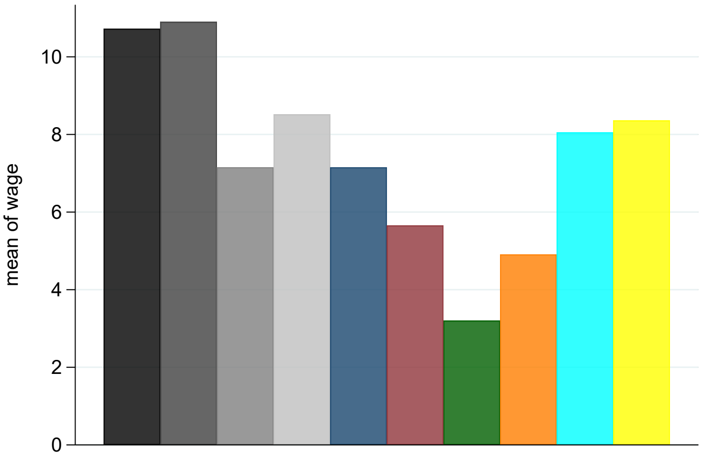
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(gs0)) bar(2, col(gs4)) bar(3, col(gs8)) bar(4, col(gs12))
bar(5, col(navy)) bar(6, col(maroon)) bar(7, col(dkgreen))
bar(8, col(orange)) bar(9, col(cyan)) bar(10, col(yellow))
```

This bar graph shows 10 bars, each displayed using a different color. By specifying `color(gs0)`, I have made the first bar black, and I have used the `color()` option to make the next three bars three different shades of gray. The next three bars are `navy`, `maroon`, and `dkgreen` (dark colors). The next bar is `orange`, and the final two bars are `cyan` and `yellow` (two bright colors). *Uses nlsw.dta*



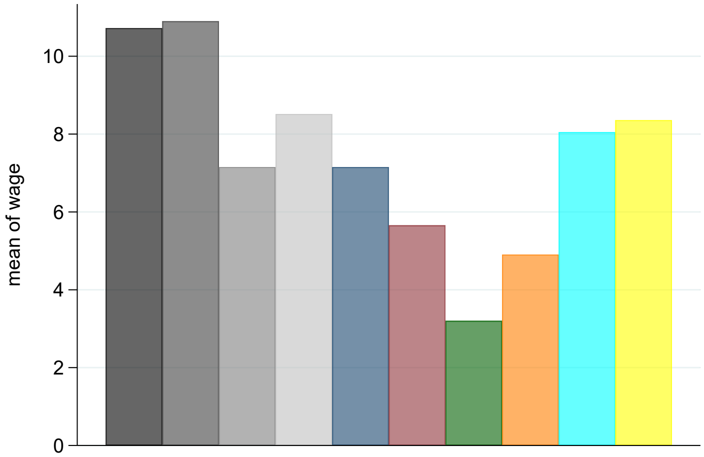
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(gs0%80)) bar(2, col(gs4%80)) bar(3, col(gs8%80)) bar(4,
col(gs12%80)) bar(5, col(navy%80)) bar(6, col(maroon%80))
bar(7, col(dkgreen%80)) bar(8, col(orange%80)) bar(9,
col(cyan%80)) bar(10, col(yellow%80))
```

This graph is identical to the prior graph, except that I have modified the opacity for each of the bars. For example, the color for the fifth bar is specified as `color(navy%80)`, displaying that bar using `navy` but using 80% opacity (or, if you prefer, 20% transparency). A little bit of the background shows through. I cannot quite see the grid lines through bars. *Uses nlsw.dta*



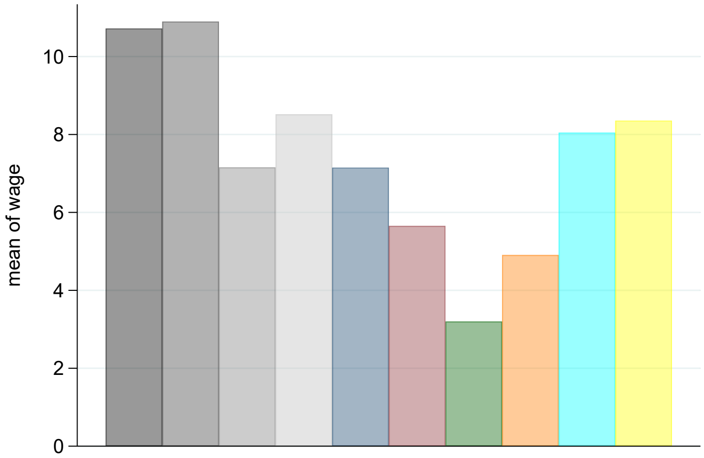
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(gs0%60)) bar(2, col(gs4%60)) bar(3, col(gs8%60)) bar(4,
col(gs12%60)) bar(5, col(navy%60)) bar(6, col(maroon%60))
bar(7, col(dkgreen%60)) bar(8, col(orange%60)) bar(9,
col(cyan%60)) bar(10, col(yellow%60))
```

This graph is identical to the prior graph, except now the opacity is specified as `60%`. I can see much more of the background showing through the bars, and I can faintly see the grid lines showing through. *Uses nlsw.dta*



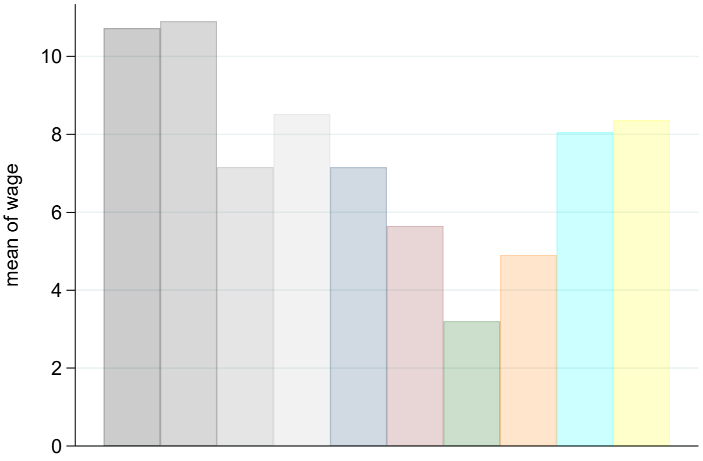
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(gs0%40)) bar(2, col(gs4%40)) bar(3, col(gs8%40)) bar(4,
col(gs12%40)) bar(5, col(navy%40)) bar(6, col(maroon%40))
bar(7, col(dkgreen%40)) bar(8, col(orange%40)) bar(9,
col(cyan%40)) bar(10, col(yellow%40))
```

This graph shows what these colors look like when they are displayed using 40% opacity. The grid lines from the background clearly show through the bars because of their increased transparency. *Uses nlsw.dta*



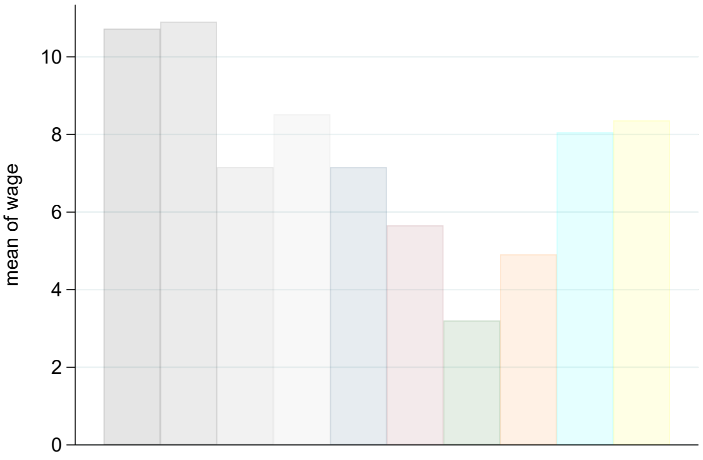
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(gs0%20)) bar(2, col(gs4%20)) bar(3, col(gs8%20)) bar(4,
col(gs12%20)) bar(5, col(navy%20)) bar(6, col(maroon%20))
bar(7, col(dkgreen%20)) bar(8, col(orange%20)) bar(9,
col(cyan%20)) bar(10, col(yellow%20))
```

In this example, the bars are now displayed with 20% opacity. The grid lines from the background come through very clearly. Also, I notice that the first four bars, showing different shades of gray, are becoming difficult to distinguish. *Uses nlsw.dta*



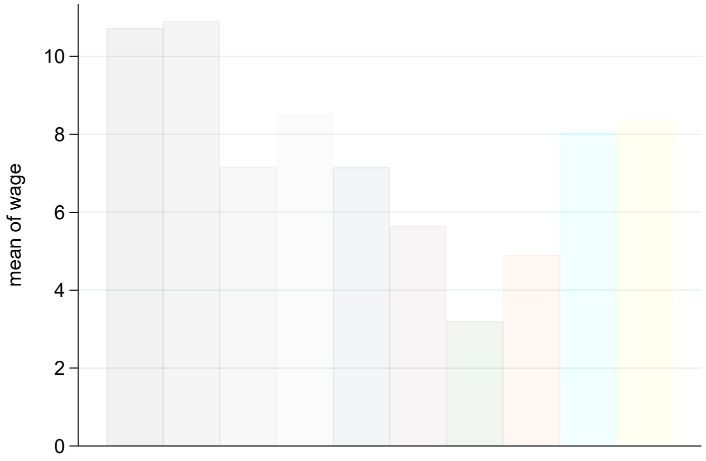
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(gs0%10)) bar(2, col(gs4%10)) bar(3, col(gs8%10)) bar(4,
col(gs12%10)) bar(5, col(navy%10)) bar(6, col(maroon%10))
bar(7, col(dkgreen%10)) bar(8, col(orange%10)) bar(9,
col(cyan%10)) bar(10, col(yellow%10))
```

This illustrates how these colors look when displayed using 10% opacity. The background grid lines are very prominent. The bars barely obscure the grid lines at all. *Uses nlsw.dta*



```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(gs0%5)) bar(2, col(gs4%5)) bar(3, col(gs8%5)) bar(4,
col(gs12%5)) bar(5, col(navy%5)) bar(6, col(maroon%5)) bar(7,
col(dkgreen%5)) bar(8, col(orange%5)) bar(9, col(cyan%5))
bar(10, col(yellow%5))
```

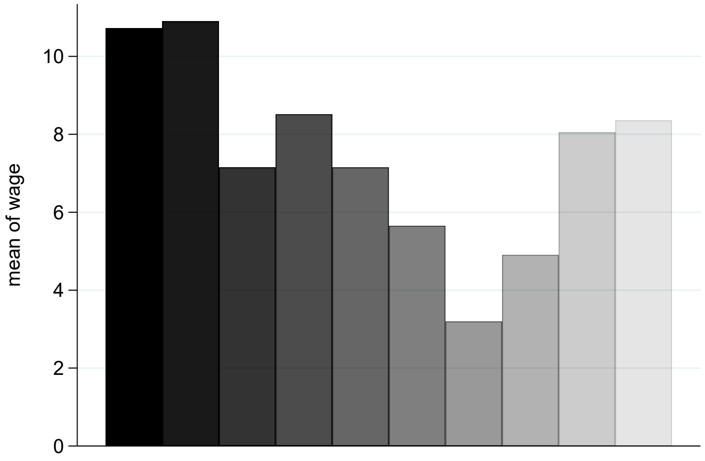
Finally, this example shows these colors displayed using 5% opacity. The colors are very faint but still perceptible (barely). *Uses nlsw.dta*



I hope these examples have given you a sense of what these 10 colors look like at different levels of opacity. At this point, I would emphasize that while I am using `graph bar` as a tool for showing you how these colors appear, you can use these colors using different kinds of graphs in a variety of contexts. For example, although displaying a bar chart using 5% opacity (95% transparency) may not be advantageous, this level of transparency can be very useful for a scatterplot with many overlapping observations (as illustrated in the examples starting in section 2.1). In the next examples, I will show you 5 bar charts, each using a single color, shown with 10 bars, where each bar is displayed using opacity that ranges from 100% down to 10%. This will allow you to see, side by side, how these colors look at differing levels of opacity. The examples will show the colors `black`, `navy`, `maroon`, `dkgreen`, and `orange`.

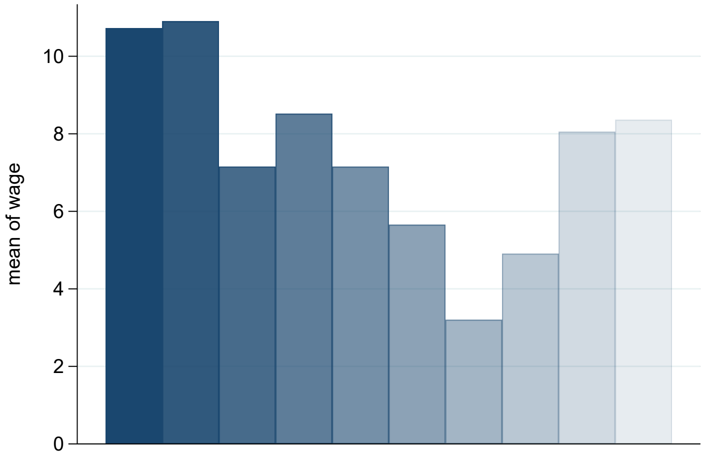
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(black%100)) bar(2, col(black%90)) bar(3, col(black%80))
bar(4, col(black%70)) bar(5, col(black%60)) bar(6,
col(black%50)) bar(7, col(black%40)) bar(8, col(black%30))
bar(9, col(black%20)) bar(10, col(black%10))
```

This graph uses the `graph bar` command to show 10 bars, each displayed using the color `black` but with opacity values ranging from 100% to 10% in 10% increments. *Uses nlsw.dta*



```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(navy%100)) bar(2, col(navy%90)) bar(3, col(navy%80))
bar(4, col(navy%70)) bar(5, col(navy%60)) bar(6,
col(navy%50)) bar(7, col(navy%40)) bar(8, col(navy%30))
bar(9, col(navy%20)) bar(10, col(navy%10))
```

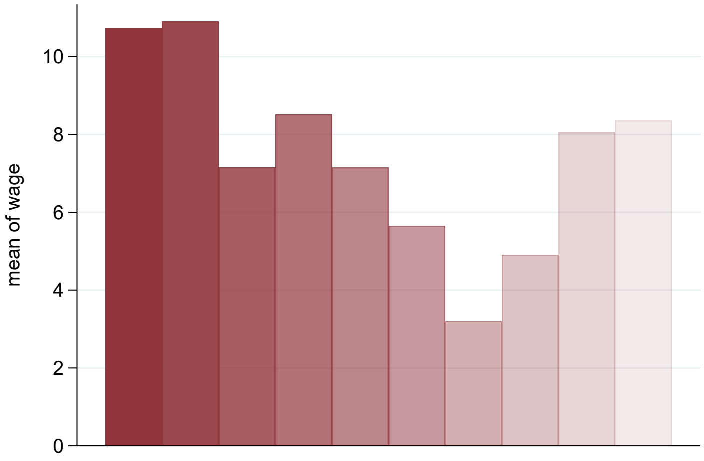
This graph uses the `graph bar` command to show 10 bars, each displayed using the color `navy` but with opacity values ranging from 100% to 10% in 10% increments. *Uses nlsw.dta*



```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(maroon%100)) bar(2, col(maroon%90)) bar(3,
col(maroon%80)) bar(4, col(maroon%70)) bar(5, col(maroon%60))
bar(6, col(maroon%50)) bar(7, col(maroon%40)) bar(8,
col(maroon%30)) bar(9, col(maroon%20)) bar(10,
col(maroon%10))
```

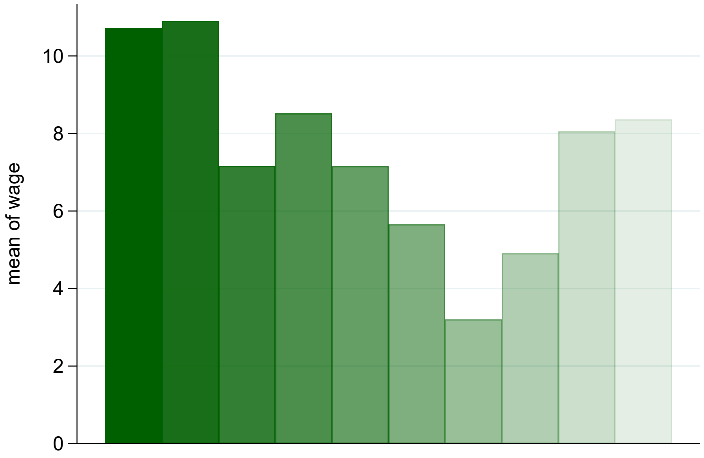
This graph uses the `graph bar` command to show 10 bars, each displayed using the color `maroon` but with opacity values ranging from 100% to 10%

in 10% increments. *Uses nlsw.dta*



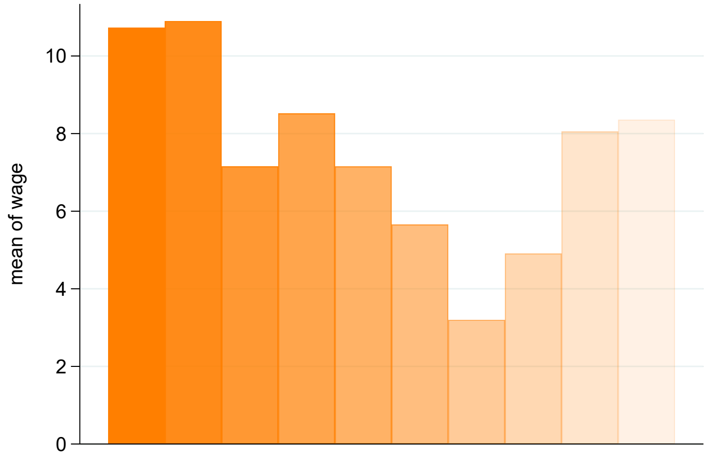
```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(dkgreen%100)) bar(2, col(dkgreen%90)) bar(3,
col(dkgreen%80)) bar(4, col(dkgreen%70)) bar(5,
col(dkgreen%60)) bar(6, col(dkgreen%50)) bar(7,
col(dkgreen%40)) bar(8, col(dkgreen%30)) bar(9,
col(dkgreen%20)) bar(10, col(dkgreen%10))
```

This graph uses the `graph bar` command to show 10 bars, each displayed using the color `dkgreen` but with opacity values ranging from 100% to 10% in 10% increments. *Uses nlsw.dta*



```
graph bar wage, over(occ10) asyvars legend(off) bar(1,
col(orange%100)) bar(2, col(orange%90)) bar(3,
col(orange%80)) bar(4, col(orange%70)) bar(5, col(orange%60))
bar(6, col(orange%50)) bar(7, col(orange%40)) bar(8,
col(orange%30)) bar(9, col(orange%20)) bar(10,
col(orange%10))
```

This graph uses the `graph bar` command to show 10 bars, each displayed using the color `orange` but with opacity values ranging from 100% to 10% in 10% increments. *Uses nlsw.dta*



10.2.4 Overlapping colors

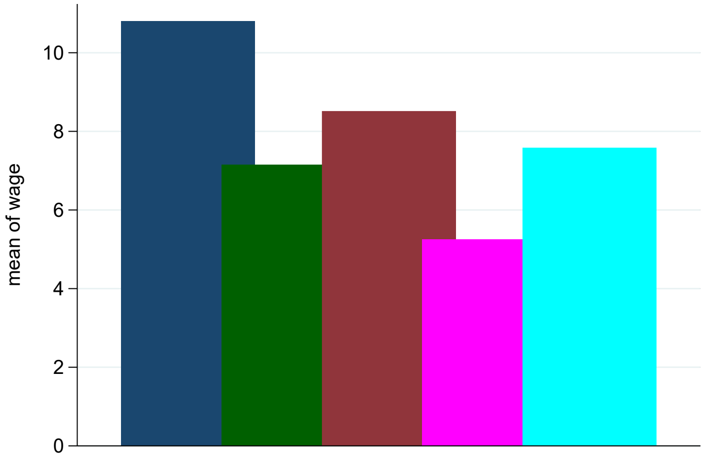
In each of the examples so far, each bar has been displayed without any overlap. In many instances, the greatest utility of displaying colors with reduced opacity (that is, increased transparency) is the ability to perceive overlapping colors. Examples include

1. displaying two overlapping distributions using a histogram (as illustrated in section 2.8).
2. overlapping distributions in a kernel density plot (as illustrated in section 2.8).
3. displaying overlapping statistical distributions (such as a normal distribution) (as illustrated in section 2.8).
4. overlapping confidence intervals using the `marginsplot` command (as illustrated in section 11.3).
5. confidence intervals overlapping a reference line using the `marginsplot` command (as illustrated in section 11.3).
6. overlapping markers (from the same variable) in a scatterplot (as illustrated in section 2.1).
7. overlapping markers (from different variables) in a scatterplot (as illustrated in section 2.11).
8. the display of a scatterplot and regression line with a confidence interval, showing the markers that fall within the confidence region (as illustrated in section 2.11).

9. the display of two (or more) groups showing overlapping scatterplots, regression lines, and confidence regions (as illustrated in section 2.3).
10. graphing bar charts with overlapping bars (as illustrated in this section; see below.)

Here I will illustrate the principles of overlapping colors, with a specific focus on 1) the order in which objects are drawn and 2) the impact of opacity on overlapping colors. I will illustrate overlap of colors using the `graph bar` command, where `gap()` is used with negative values to create overlap among bars.

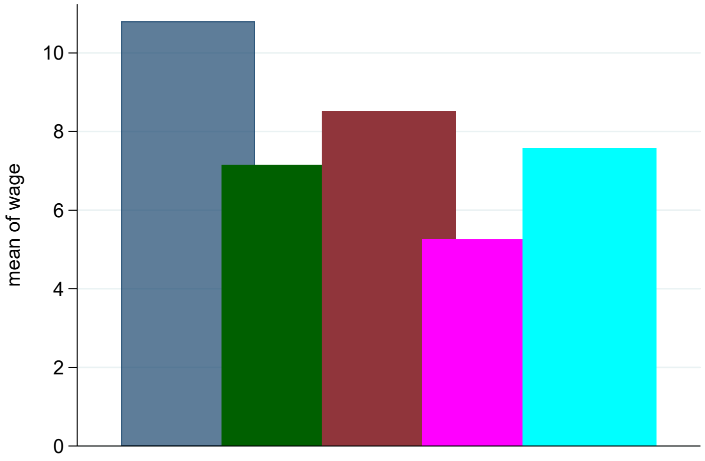
```
graph bar wage, over(occ5, gap(-25)) asyvars legend(off)
bar(1, color(navy)) bar(2, color(dkgreen)) bar(3,
color(maroon)) bar(4, color(magenta)) bar(5, color(cyan))
```



This `graph bar` command uses the `gap(-25)` option to create overlapping bars. Note how the second bar (in green) overlaps the first bar (in blue). The bars are drawn in order, drawing the first bar, then the second bar, and so forth. The second bar uses the default opacity (no transparency), so the portion where it overlaps the first bar is shown in green and fully obscures the blue bar. *Uses nlsw.dta*

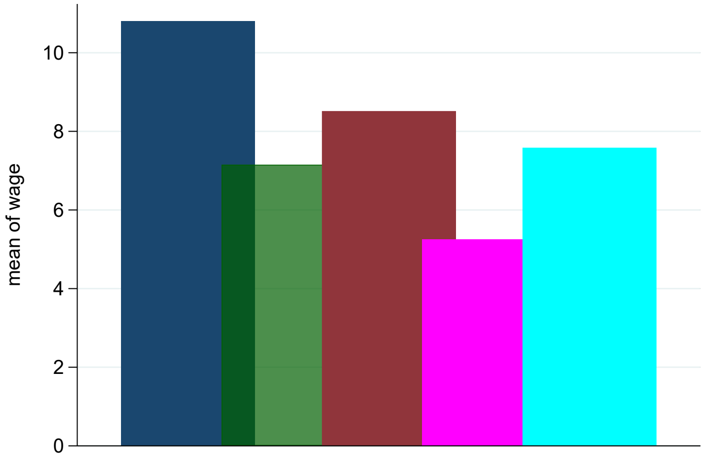
```
graph bar wage, over(occ5, gap(-25)) asyvars legend(off)
bar(1, color(navy%70)) bar(2, color(dkgreen)) bar(3,
color(maroon)) bar(4, color(magenta)) bar(5, color(cyan))
```

Let's now reduce the opacity of the first bar by specifying the option `bar(1, color(navy%70))`. The bar is now somewhat transparent; that is, we can barely see the grid lines showing through that bar. The navy bar was drawn first, then the green bar was drawn, and so on. Notice that the second bar uses the default opacity (no transparency), so no color from the first bar shows through the second bar. *Uses nlsw.dta*



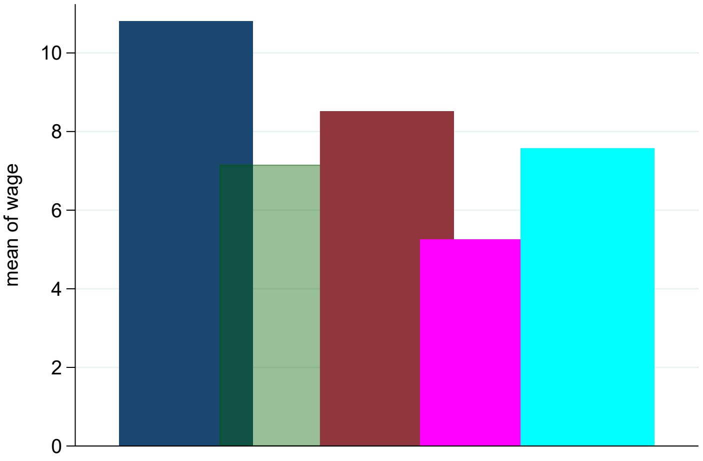
```
graph bar wage, over(occ5, gap(-25)) asyvars legend(off)
bar(1, color(navy)) bar(2, color(dkgreen%70)) bar(3,
color(maroon)) bar(4, color(magenta)) bar(5, color(cyan))
```

This example specifies the option `bar(2, color(dkgreen%70))`, displaying the second bar in dark green, with 70% opacity (30% transparency). I can now see the grid lines through the second bar. Further, where the second bar overlaps with the first, I see some of the blue showing through the green. *Uses nlsw.dta*



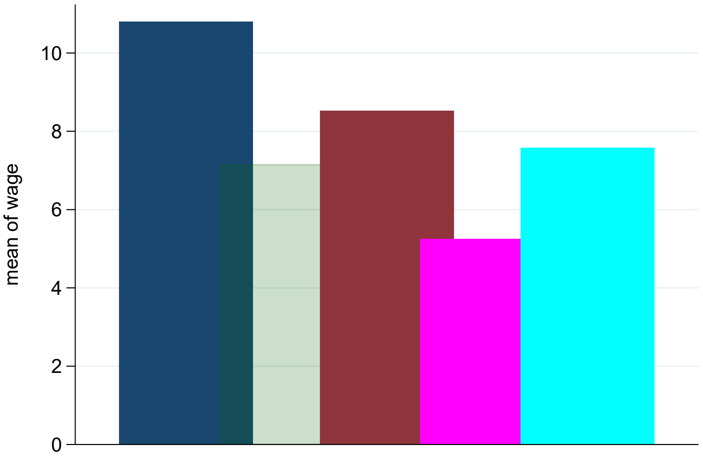
```
graph bar wage, over(occ5, gap(-25)) asyvars legend(off)
bar(1, color(navy)) bar(2, color(dkgreen%40)) bar(3,
color(maroon)) bar(4, color(magenta)) bar(5, color(cyan))
```

I have further reduced the opacity of the green bar by specifying `color(dkgreen%40)`. The green bar is now more transparent, showing the grid lines from the background more clearly. Where the first and second bars overlap, more of the blue bar is allowed to show through. In fact, the overlap looks like a mixture of blue and green, where in the prior example, the overlap was mostly green. *Uses nlsw.dta*



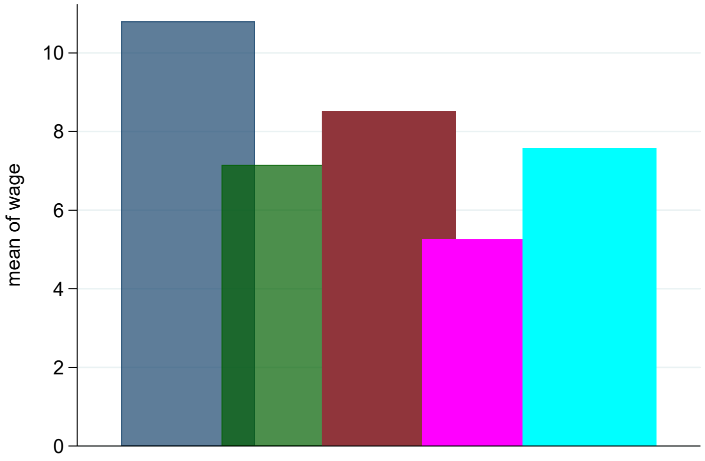
```
graph bar wage, over(occ5, gap(-25)) asyvars legend(off)
bar(1, color(navy)) bar(2, color(dkgreen%20)) bar(3,
color(maroon)) bar(4, color(magenta)) bar(5, color(cyan))
```

Now I have dramatically reduced the opacity of the second bar by specifying `color(dkgreen%20)`. The green bar is now very transparent. The grid lines show through very clearly. Further, where the first and second bars overlap, so much blue is allowed to show through that it is hard to see any of the green. *Uses nlsw.dta*



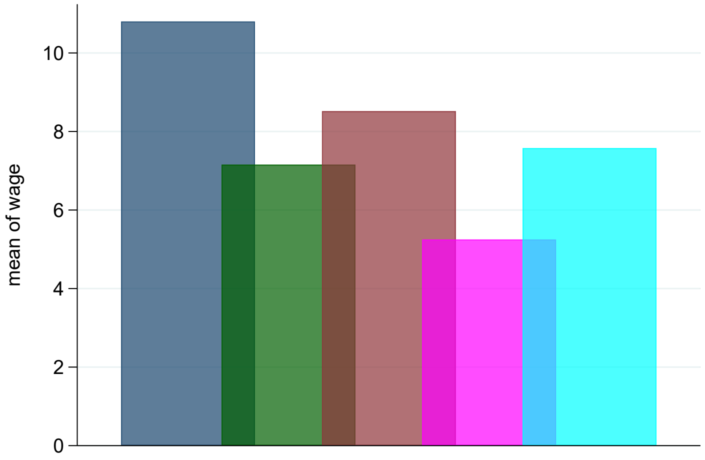
The prior examples explored what happens when we manipulate the opacity of one of the bars. Now let’s see what happens when we manipulate the opacity of the first two bars.

```
graph bar wage, over(occ5, gap(-25)) asyvars legend(off)
bar(1, color(navy%70)) bar(2, color(dkgreen%70)) bar(3,
color(maroon)) bar(4, color(magenta)) bar(5, color(cyan))
```



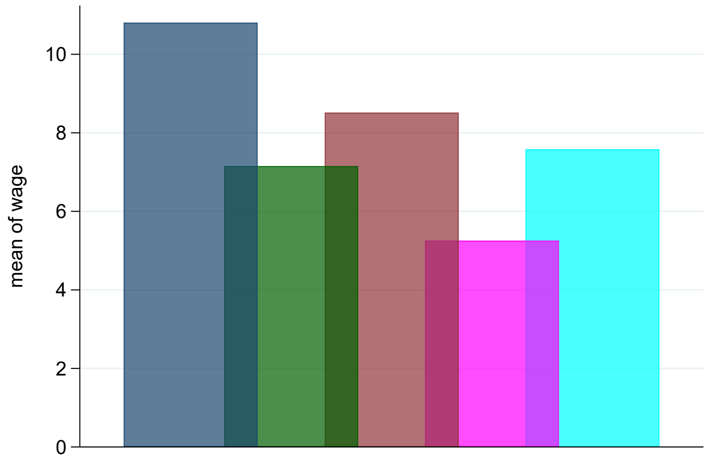
In this example, I have specified the opacity for the first and second bar both to be 70% (that is, having 30% transparency). For each bar, we can see the grid lines showing through the bars. And, where the bars overlap, both bars are visible. The color of this section is mainly green, with some blue showing through. I like this look. In the next graph, I will apply the same level of opacity to all the bars. *Uses nlsw.dta*

```
graph bar wage, over(occ5, gap(-25)) asyvars legend(off)
bar(1, col(navy%70)) bar(2, col(dkgreen%70)) bar(3,
col(maroon%70)) bar(4, col(magenta%70)) bar(5, col(cyan%70))
```



Each bar is displayed using 70% opacity. I can faintly see the grid lines behind each bar. Where the first and second bars overlap, the color of the second bar is most prominent, but some of the color from the first bar is allowed to show through. All the overlapping regions are like this. For example, the overlap of the fourth and fifth bars is mostly cyan but some magenta is allowed to show through. *Uses nlsw.dta*

```
graph bar wage, over(occ5, gap(-25) reverse descending)
asyvars legend(off) bar(1, col(navy%70)) bar(2,
col(dkgreen%70)) bar(3, col(maroon%70)) bar(4,
col(magenta%70)) bar(5, col(cyan%70))
```



Suppose I like the prior graph but I want the bars drawn in reverse order, so the fourth bar overlaps the fifth, the third bar overlaps the fourth, and so forth. Using the `reverse descending` options (illustrated in section 4.4), I draw the bars in reverse order (5 then 4 then 3 then 2 then 1). As a result, note how the overlap of bars 4 and 5 is now dominated by magenta, with a little bit of `cyan` showing through. *Uses nlsw.dta*

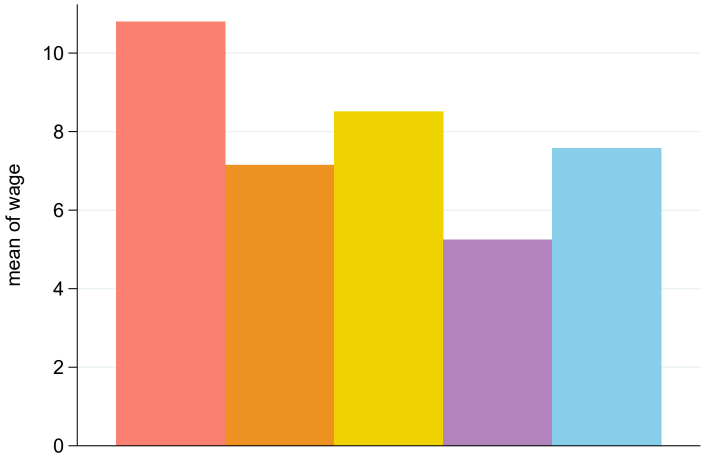
10.2.5 Specifying colors using RGB, CMYK, and HSV values

The next section illustrates other, more technical, methods that you can use for specifying colors—namely, by specifying RGB values, specifying CMYK values, and specifying HSV values. I am going to assume that if you are choosing to specify colors using one of these methods, you understand the technical details about specifying colors, and you just need to know how you can do so in Stata. In the following examples, I will illustrate how to make a bar graph with five bars, using five colors that I found documented on Wikipedia. The five colors are *Salmon*, *Carrot Orange*, *Safety Yellow*, *African Violet*, and *Sky Blue*. Here are the specific links I used from Wikipedia to obtain the values I use to specify these colors using RGB, CMYK, and HSV values.

* *Salmon*: https://en.wikipedia.org/wiki/Salmon_(color)
* *Carrot Orange*: https://en.wikipedia.org/wiki/Shades_of_orange#Carrot_orange
* *Safety Yellow*: https://en.wikipedia.org/wiki/Shades_of_yellow#Safety_yellow

* *African Violet*: https://en.wikipedia.org/wiki/Shades_of_violet#African_violet
* *Sky Blue*: https://en.wikipedia.org/wiki/Sky_blue

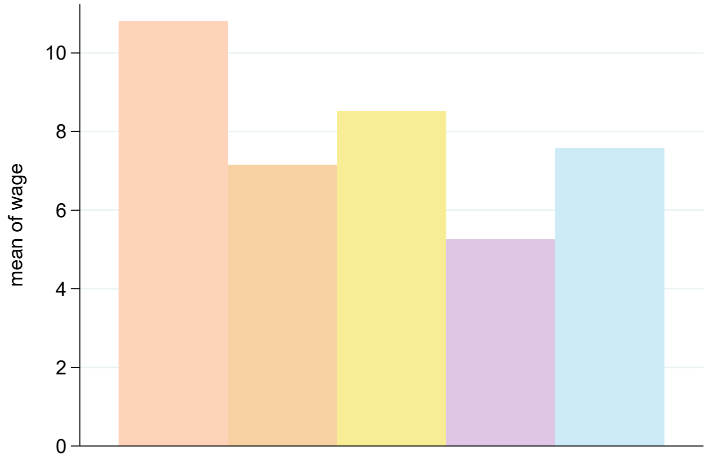
```
graph bar wage, over(occ5) asyvars legend(off) bar(1,
color("250 128 114")) bar(2, color("237 145 33")) bar(3,
color("238 210 2")) bar(4, color("178 132 190")) bar(5,
color("135 206 235"))
```



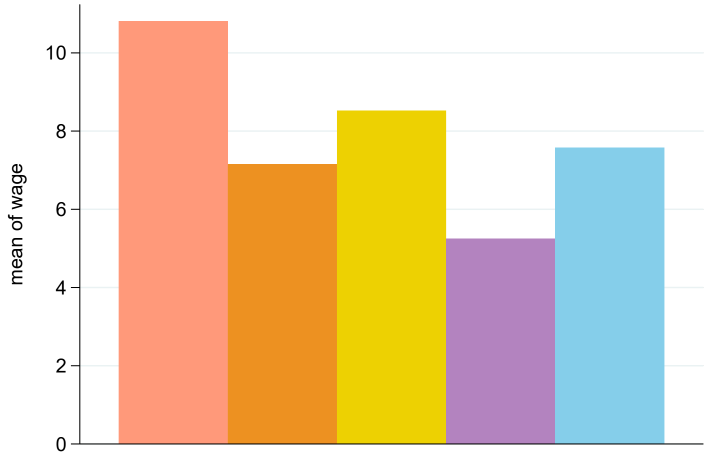
This example uses RGB colors to specify the colors of the five bars. The first bar, shown in *Salmon*, specifies the RGB values using the option `color("250 128 114")`. The three colors are given, specified in quotation marks. The second bar uses the RGB values for *Carrot orange*, the third using the RGB values for *Safety Yellow*, the fourth using *African Violet*, and the fifth using *Sky Blue*. *Uses nlsw.dta*

```
graph bar wage, over(occ5) asyvars legend(off) bar(1,
color("0 45 69 0")) bar(2, color("0 39 86 7")) bar(3,
color("0 12 99 7")) bar(4, color("6 31 0 26")) bar(5,
color("43 12 0 8"))
```

This example illustrates creation of the same graph as the prior example, but the colors are specified using CMYK values. Using the option `color("0 45 69 0")` for the first bar specifies that bar will be displayed using the color *Salmon* based on the four specified CMYK values. The `color()` option assumes that when four numbers are specified in quotes, those refer to CMYK values. *Uses nlsw.dta*



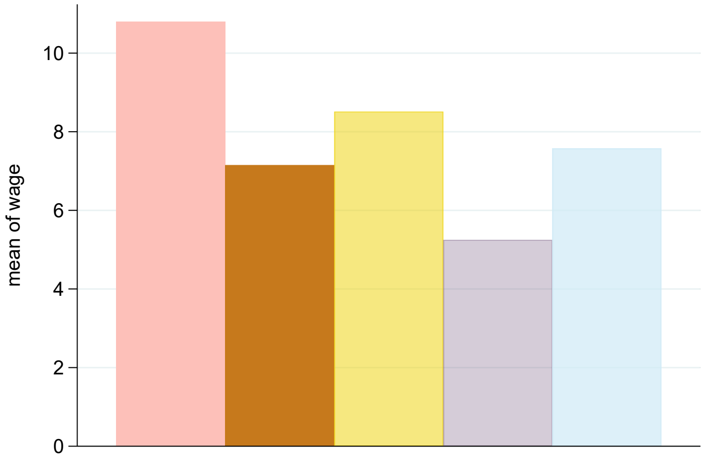
```
graph bar wage, over(occ5) asyvars legend(off) bar(1,
col("hsv 14 .52 1.00")) bar(2, col("hsv 33 0.86 0.93"))
bar(3, col("hsv 53 0.99 0.93")) bar(4, col("hsv 288 0.31
0.75")) bar(5, col("hsv 197 0.43 0.92"))
```



This example creates the same graph as the prior example, but the colors are specified using HSV values. The first bar specifies the color *Salmon* using HSV values by specifying `color("hsv 14 .52 1.00")`. Because RGB and HSV values both use a triplet of numbers to specify the color, Stata requires HSV values to be prefaced with `hsv`. *Uses nlsw.dta*

You can apply an intensity modifier (that is, `*#`) to colors specified using RGB, CMYK, and HSV. You can also specify opacity for such colors. I am going to repeat the three examples from above, with the following modifications. The first bar will use an intensity modifier of `*0.5` to make the *Salmon* color brighter. The second bar will use an intensity modifier of `*1.2` to make the *Carrot Orange* color darker. The *Safety Yellow* color of the third bar will be displayed with less opacity (more transparency) by specifying `%50`. The *African Violet* color of the fourth bar will be modified in both its intensity and opacity by specifying `*1.5%30` to both increase its brightness and reduce its opacity. The *Sky Blue* color of the fifth bar will be modified in both its intensity and opacity by specifying `*0.4%70` to both decrease its brightness and reduce its opacity.

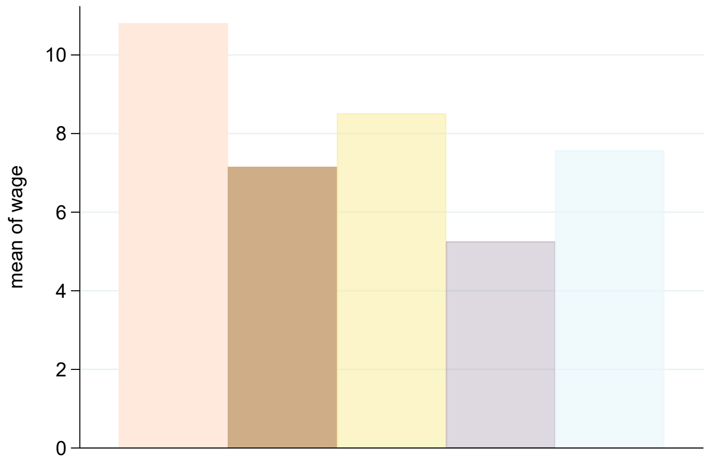
```
graph bar wage, over(occ5) asyvars legend(off) bar(1,
col("250 128 114*0.5")) bar(2, col("237 145 33*1.2")) bar(3,
col("238 210 2%50")) bar(4, col("178 132 190*1.5%30")) bar(5,
col("135 206 235*0.4%70"))
```



This example is identical to the prior example that illustrated RGB colors but illustrates the specification of intensity and opacity. Note the intensity multiplier of `*0.5` and `*1.2` on first and second bars. The third bar is shown with less opacity, `%50`, while the fourth bar uses `*1.5%30` to both increase its brightness and reduce its opacity, while the fifth bar uses `*0.4%70` to both decrease its brightness and reduce its opacity. *Uses nlsw.dta*

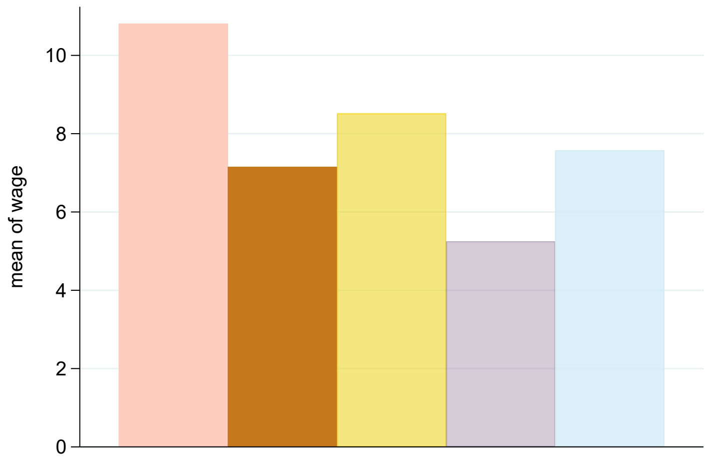
```
graph bar wage, over(occ5) asyvars legend(off) bar(1, col("0
45 69 0*0.5")) bar(2, col("0 39 86 7*1.2")) bar(3, col("0 12
99 7%50")) bar(4, col("6 31 0 26*1.5%30")) bar(5, col("43 12
0 8*0.4%70"))
```

This example is identical to the prior example that illustrated CMYK colors, but it also illustrates the same adjustments to intensity and opacity from the prior example. The adjustments to intensity and opacity are applied in the exact same way for this example (using CMYK colors) as they are for RGB colors. *Uses nlsw.dta*



```
graph bar wage, over(occ5) asyvars legend(off) bar(1,
col("hsv 14 .52 1.00*0.5")) bar(2, col("hsv 33 0.86
0.93*1.2")) bar(3, col("hsv 53 0.99 0.93%50")) bar(4,
col("hsv 288 0.31 0.75*1.5%30")) bar(5, col("hsv 197 0.43
0.92*0.4%70"))
```

This example is identical to the prior example that illustrated HSV colors, but it also illustrates the same adjustments to intensity and opacity from the prior example. The adjustments to intensity and opacity are applied in the exact same way for this example (using CMYK colors) as they are for RGB colors. *Uses nlsw.dta*



To recap, this section has illustrated ways you can control the colors/intensity/opacity of graph elements. You can see `help colorstyle` for more details.

If you are hungering for even more information about how to control the color of graph elements, the Internet has many resources that you might enjoy. When I searched the Internet using keywords such as “Stata graph colors", I found many interesting resources. In particular, I would call your attention to a suite of commands created by Ben Jann called *palettes*. Here are three ways you can learn more about that suite of commands:

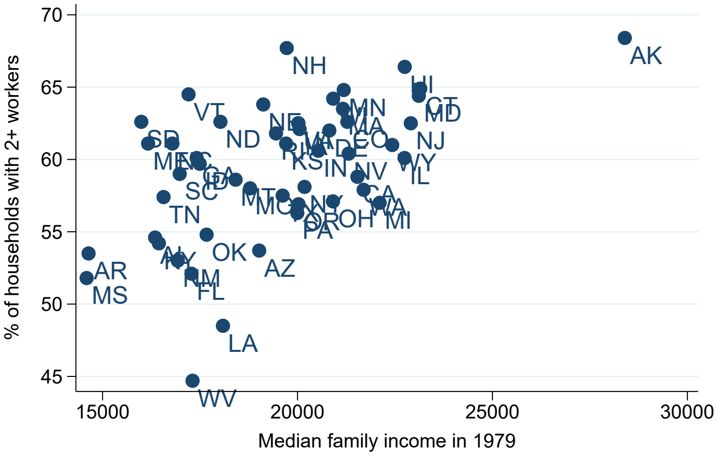
- Jann, B. 2018. Color palettes for Stata graphics. *Stata Journal* 18: 765–785. https://doi.org/10.1177/1536867X1801800402.
- Jann, B. 2018. Color palettes for Stata graphics. University of Bern Social Sciences Working Papers No. 31. https://ideas.repec.org/p/bss/wpaper/31.html.
- Jann, B. 2017. PALETTES: Stata module providing color palettes, symbol palettes, and line pattern palettes. https://ideas.repec.org/c/boc/bocode/s458444.html.

10.3 Clock position

A clock position refers to a location using the numbers on an analog clock to indicate the location, with twelve o'clock being above the center, three o'clock to the right, six o'clock below the center, and nine o'clock to the left. A value of 0 refers to the center but may not always be valid. See `help clockposstyle` for more information.

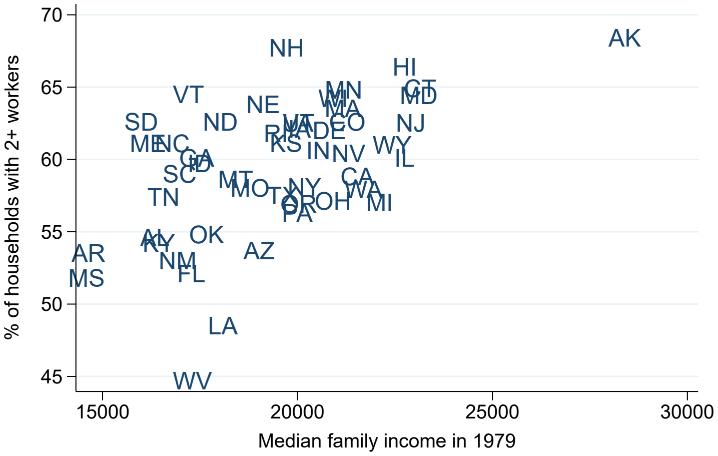
```
scatter workers2 faminc, mlabel(stateab) mlabposition(5)
```

In this example, we add marker labels to a scatterplot and use the `mlabposition(5)` (marker label position) option to place the marker labels in the five o'clock position with respect to the markers. *Uses allstatesdc.dta*



```
scatter workers2 faminc, mlabel(stateab) mlabposition(0)
msymbol(i)
```

Here we place the markers in the center position by using the `mlabposition(0)` option. We also make the symbols invisible by using the `msymbol(i)` option. Otherwise, the markers and marker labels would be atop each other. *Uses allstatesdc.dta*

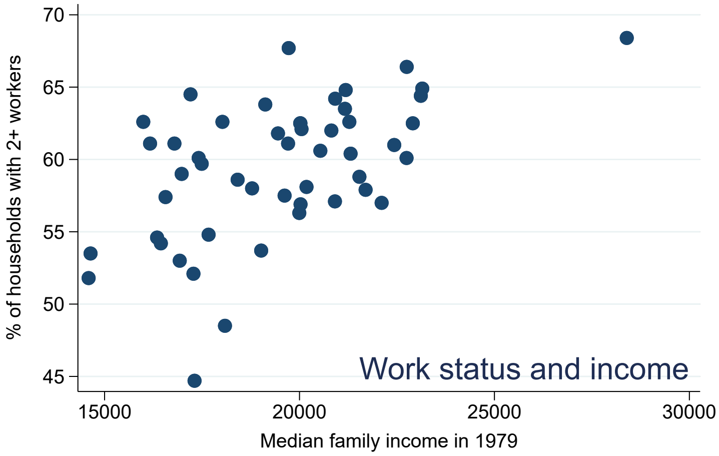

% of households with 2+ workers
70
65
60
55
50
45
15000
20000
25000
30000
Median family income in 1979
AK
NH
HI
CT
MN
MD
VT
NE
WI
MA
SD
ND
NJ
UT
VA
DE
CO
RI
KS
ME
NC
WY
IN
NV
GA
ID
IL
SC
CA
MT
MO
NY
TX
WA
OR
OH
MI
TN
PA
OK
AL
KY
AR
AZ
NM
FL
MS
LA
WV

10.4 Compass direction

A *compassdirstyle* is much like *clockpos*, but where a *clockpos* has 12 possible outer positions, like a clock, the *compassdirstyle* has only 9 possible outer positions, like the major labels on a compass: `north`, `neast`, `east`, `seast`, `south`, `swest`, `west`, `nwest`, and `center`. These can be abbreviated as `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw`, and `c`. Stata permits you to use a *clockpos* even when a *compassdirstyle* is called for and makes intuitive translations; for example, `12` is translated to `north`, or `2` is translated to `neast`. See `help compassdirstyle` for more information.

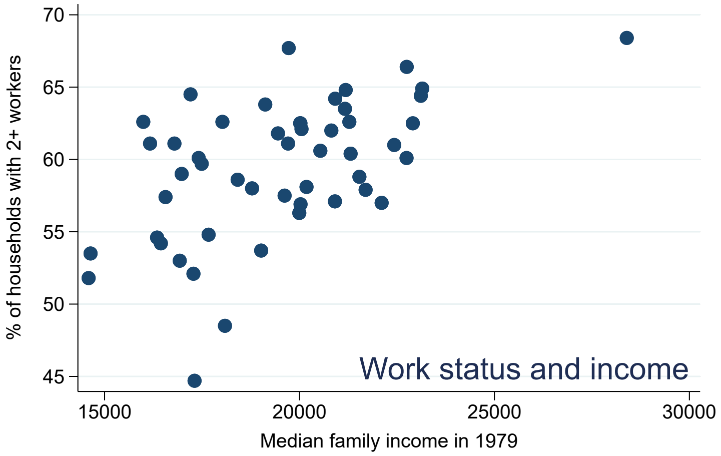
```
scatter workers2 faminc, title("Work status and income",
ring(0) placement(se))
```

Here the `placement()` option positions the title in the southeast (bottom right corner) of the plot region. The `ring(0)` option moves the title inside the plot region. *Uses allstatesdc.dta*



```
scatter workers2 faminc, title("Work status and income",
ring(0) placement(4))
```

If we instead specify the `placement(4)` option (using a *clockpos* instead of *compassdir*), Stata makes a suitable substitution, and the title is placed in the bottom right corner. *Uses allstatesdc.dta*

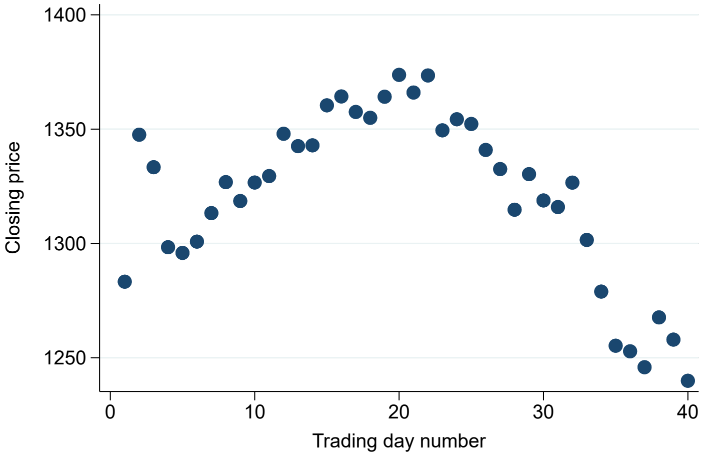

Work status and income
% of households with 2+ workers
70
65
60
55
50
45
15000
20000
25000
30000
Median family income in 1979

10.5 Connecting points

Stata supports a variety of methods for connecting points using different values for the *connectstyle*. These include `l` (lowercase L, as in line) to connect with a straight line, `L` to connect with a straight line only if the current x value is greater than the prior x value, `J` for stairstep, `stepstair` for step then stair, and `i` for invisible connections. The next few examples use the `spjanfeb2001` data file, keeping only the data for January and February of 2001. See `help connectstyle` for more information.

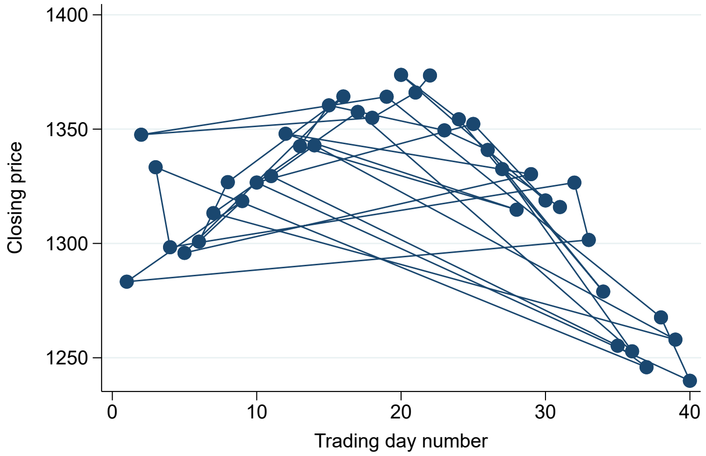
```
scatter close tradeday
```

Here we make a scatterplot showing the closing price on the y axis and the trading day (numbered 1–40) on the x axis. Normally, we would connect these points. *Uses spjanfeb2001.dta*

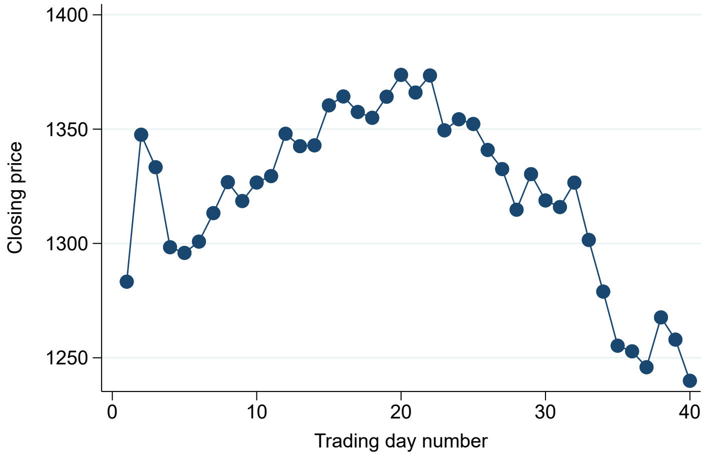


```
graph twoway connected close tradeday
```

We change to using the `connected` command to connect the points, but this is probably not the kind of graph we wanted to create. The problem is that the observations are in a random order, but the observations are connected in the same order as they appear in the data. We really want the points to be connected based on the order of `tradeday`. *Uses spjanfeb2001.dta*



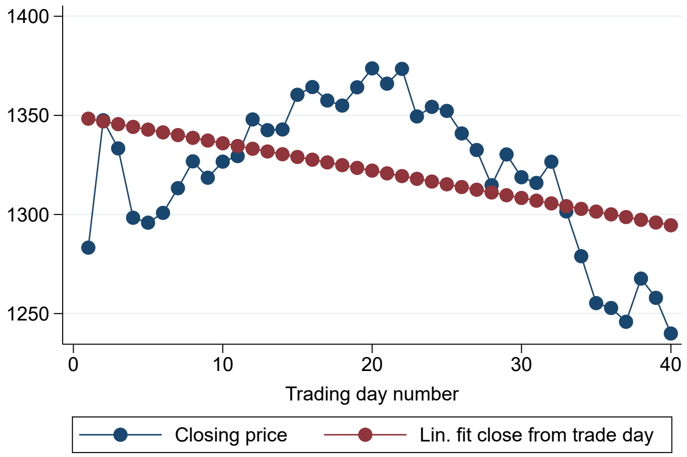
```
graph twoway connected close tradeday, sort
```



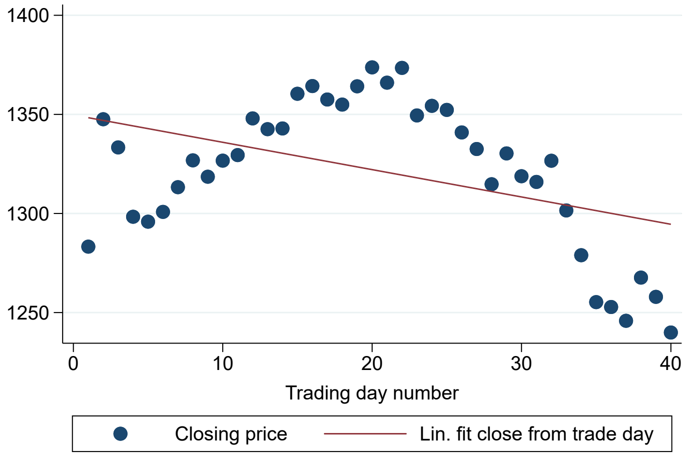
To fix the previous graph, we can either first use the `sort` command to sort the data on `tradeday` or, as we do here, use the `sort` option to tell Stata to sort the data on `tradeday` before connecting the points. We could have also specified `sort(tradeday)`, and it would have had the same effect. *Uses spjanfeb2001.dta*

```
graph twoway connected close predclose tradeday, sort
```

Say that we used the `regress` command to predict `close` from `tradeday` and generated a predicted value called `predclose`. Here we plot the actual closing prices and the predicted closing prices. *Uses spjanfeb2001.dta*



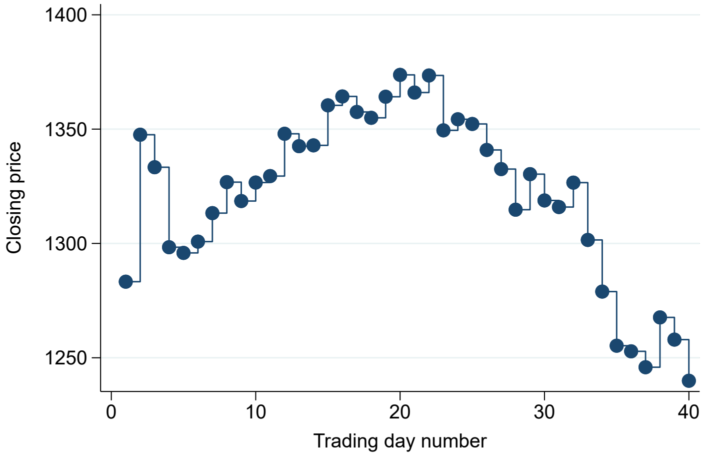
```
graph twoway connected close predclose tradeday, sort
connect(i .) msymbol(. i)
```



We use the `connect(i .)` option to suppress connecting the observed values while leaving the predicted values connected. The `i` option suppresses connecting the observed values whereas the `.` option indicates that the predicted values should be unchanged (that is, remain connected). We also add `msymbol(. i)` to make the symbols for the observed values unchanged, but invisible for the fitted values. *Uses spjanfeb2001.dta*

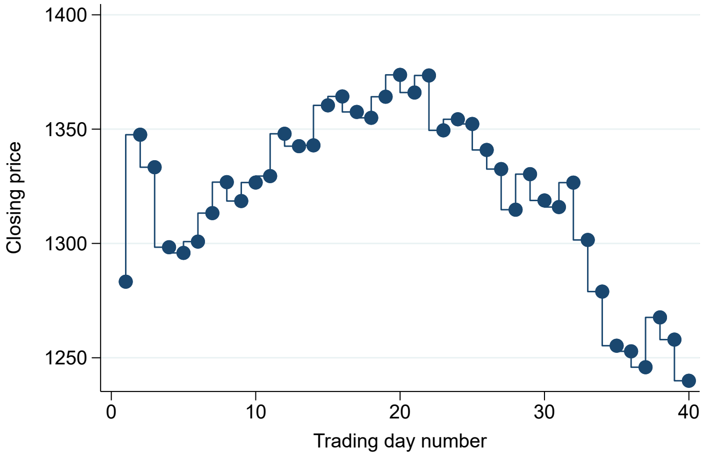
```
graph twoway connected close tradeday, connect(J) sort
```

In other contexts (such as survival analysis), we might want to connect points using a stairstep pattern. Here we connect the observed closing prices with the `J` option (which can also be specified as `stairstep`) to get a stairstep effect. *Uses spjanfeb2001.dta*



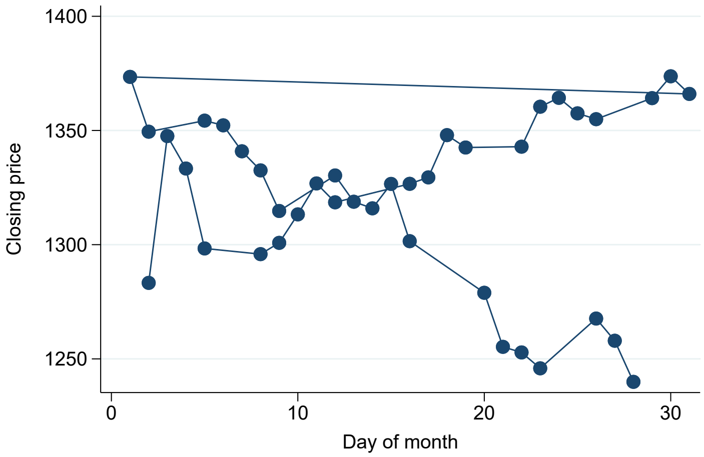
```
graph twoway connected close tradeday, connect(stepstair)
sort
```

In other contexts, we might want to connect points using a stepstair pattern. Here we connect the observed closing prices with the `stepstair` option to get a stepstair effect. *Uses spjanfeb2001.dta*



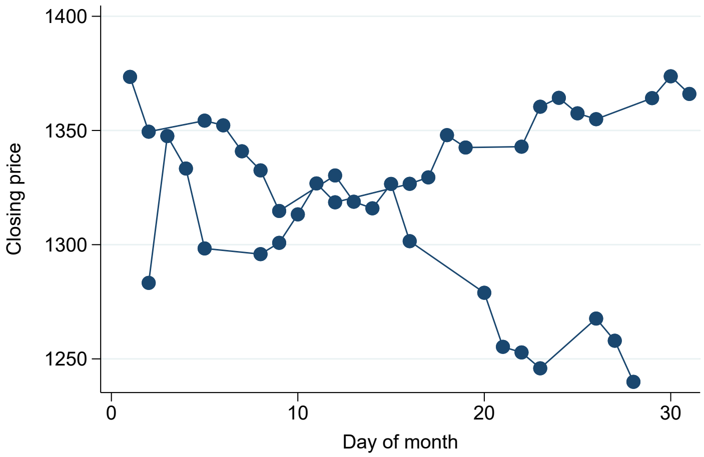
```
graph twoway connected close dom, connect(l) sort(date)
```

Say that we created a variable called `dom` that represented the day of the month and wanted to graph the closing prices for January and February against the day of the month. By using the `sort(date)` option, we almost get what we want, but there is a line that swoops back connecting January 31 to February 1. *Uses spjanfeb2001.dta*



```
graph twoway connected close dom, connect(L) sort(date)
```

This kind of example calls for the `connect(L)` option, which avoids the line that swoops back by connecting points with a straight line, except when the x value (`dom`) decreases (for example, goes from 31 to 1). *Uses spjanfeb2001.dta*

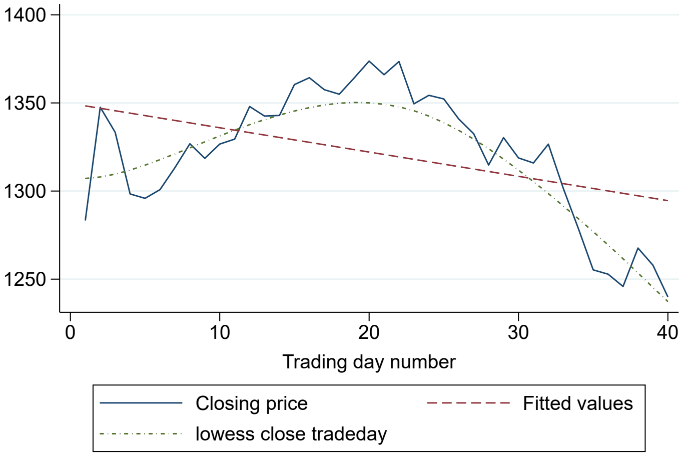


10.6 Line pattern

We can specify the pattern we want for a line in three ways. We can specify a word that selects from a set of predefined styles, including `solid` (solid line), `dash` (a dashed line), `dot` (a dotted line), `shortdash` (short dashes), `longdash` (long dashes), and `blank` (invisible). There are also the combination styles `dash_dot`, `shortdash_dot`, and `longdash_dot`. We can also use a formula that combines the following five elements in any way that we want: `l` (letter l, solid line), `_` (underscore, long dash), `-` (hyphen, medium dash), `.` (period, short dash that is almost a dot), and `#` (small amount of space). You could specify `longdash_dot` or `"_."`, and they would be equivalent. See `help linepatternstyle` for more information.

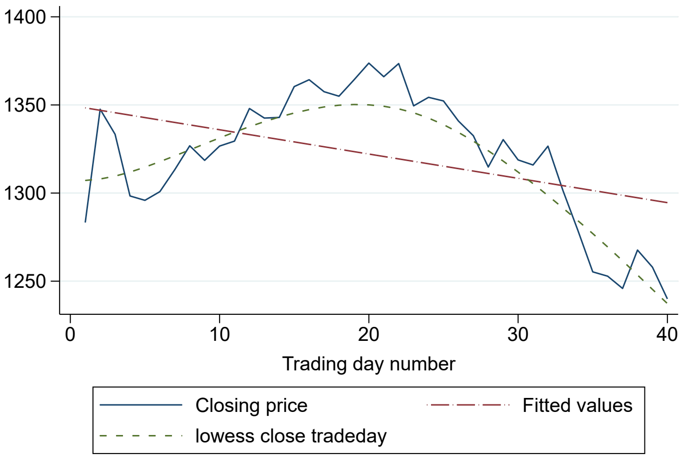
```
twoway (line close tradeday, lpattern(solid) sort) (lfit
close tradeday, lpattern(dash)) (lowess close tradeday,
lpattern(shortdash_dot))
```

Here we make a line plot and use the `lpattern()` (line pattern) option to obtain a solid pattern for the observed data, a dash for the linear fit line, and a short dash and dot line for a lowess fit. *Uses spjanfeb2001.dta*



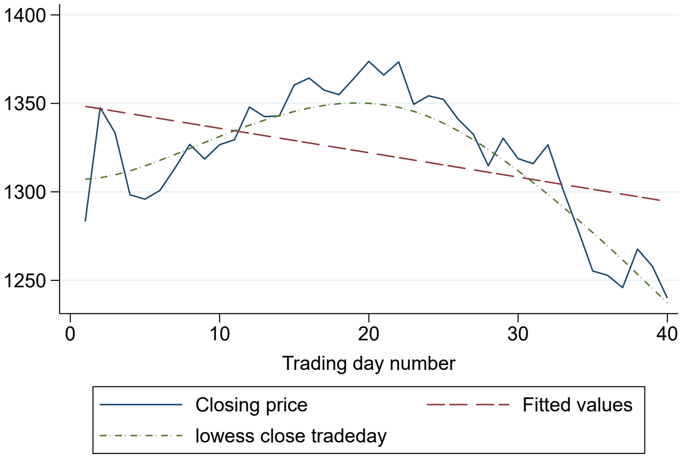
```
twoway (line close tradeday, lpattern("l") sort) (lfit close
tradeday, lpattern("._")) (lowess close tradeday, lpattern("-
###"))
```

The `lpattern()` option specifies a formula to indicate the pattern for the lines. Here we specify a solid line for the `line` plot, a dot and dash for the `lfit` plot, and a dash and three spaces for the `lowess` fit. *Uses spjanfeb2001.dta*



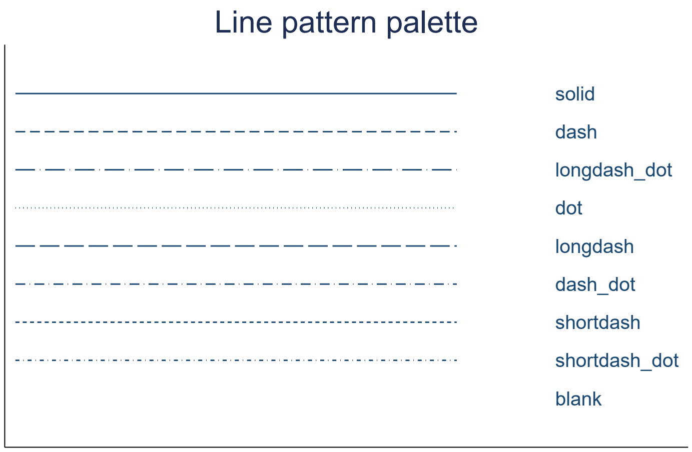
```
twoway (line close tradeday, lpattern("l") sort) (lfit close
tradeday, lpattern("__##")) (lowess close tradeday,
lpattern("-.#"))
```

This example shows other formulas to create, including `"__##"`, which yields long dashes with long breaks between, and `"-.#"`, which yields a medium dash, a dot, and a space. Using these formulas, we can create a wide variety of line patterns for those instances where we need to differentiate multiple lines. *Uses spjanfeb2001.dta*



```
palette linepalette
```

The built-in Stata command `palette linepalette` shows the line patterns that are available within Stata to help us choose a pattern to our liking. *Uses spjanfeb2001.dta*

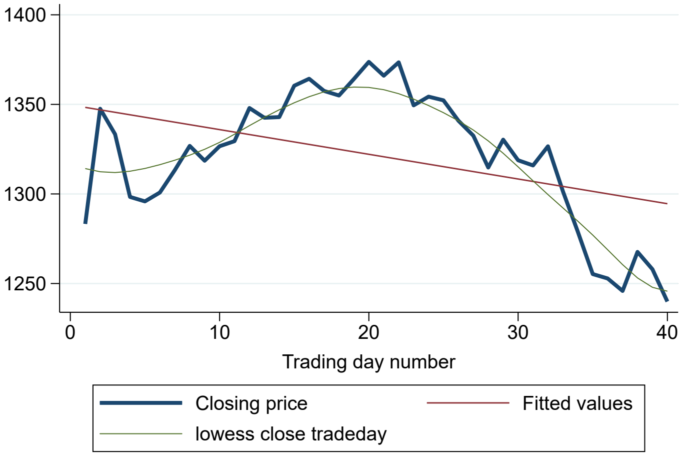

Line pattern palette
solid
dash
longdash_dot
dot
longdash
dash_dot
shortdash
shortdash_dot
blank

10.7 Line width

We can indicate the width of a line in multiple ways. I think the most common method is by specifying a keyword such as `none` (no width, invisible), `vvvthin`, `vvthin`, `vthin`, `thin`, `medthin`, `medium`, `medthick`, `thick`, `vthick`, `vvthick`, and even `vvvthick`. Also, we can specify a multiple of the line's normal thickness (for example, `*2` is twice as thick, or `*.7` is 0.7 times as thick). These units used by these methods are relative. By contrast, you can specify the line thickness using absolute units, such as points, inches, or centimeters. (If you will be resizing your graph, please see section 9.3.1, which illustrates the impact of using absolute units versus relative units when resizing graphs.) In this section, I will illustrate how to specify the width of lines using relative units (descriptive labels and multiples of the line thickness) and absolute units (using points, inches, and centimeters). See `help linewidthstyle` for more information.

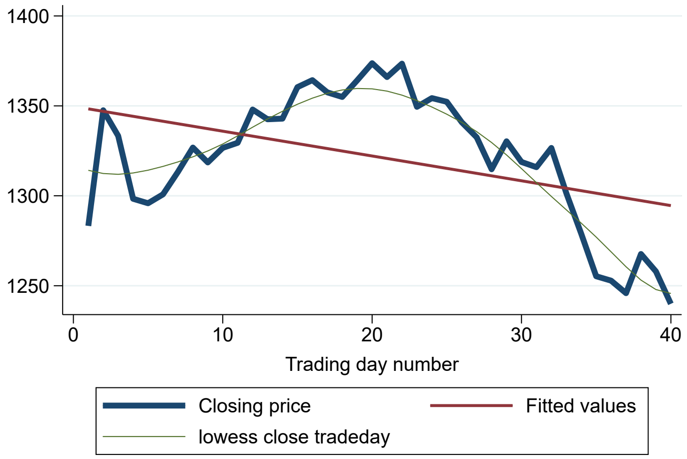
```
twoway (line close tradeday, lwidth(thick) sort) (lfit close
tradeday, lwidth(medium)) (lowess close tradeday, bwidth(.5)
lwidth(thin))
```

We now plot the same three lines from the previous section, but this time we differentiate them with line thickness by using the `lwidth()` (line width) option. *Uses spjanfeb2001.dta*



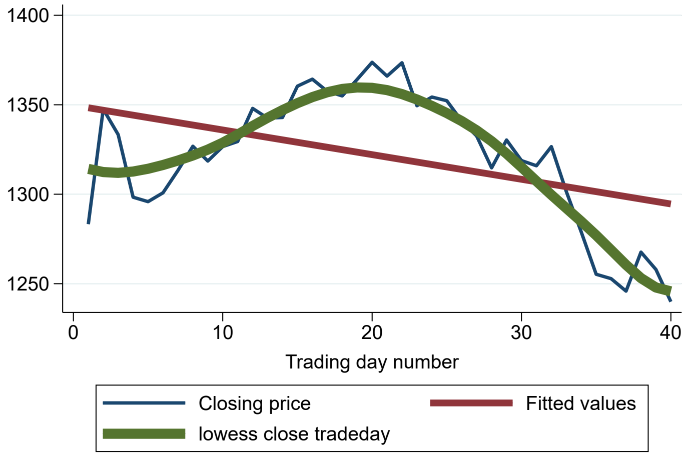
```
twoway (line close tradeday, lwidth(*4) sort) (lfit close
tradeday, lwidth(*2)) (lowess close tradeday, bwidth(.5)
lwidth(*.5))
```

This example specifies the width of the lines as a multiple of the original line thickness, making the line for the `line` plot four times as wide as the original line, the line for the `lfit` plot twice as wide, and the line for the `lowess` plot half as wide. *Uses spjanfeb2001.dta*



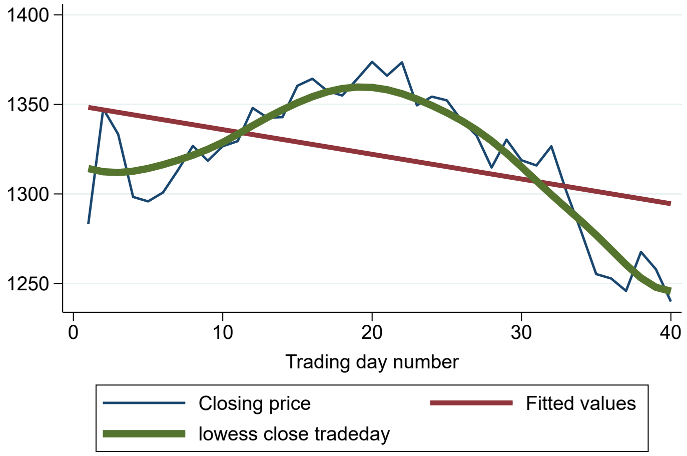
```
twoway (line close tradeday, lwidth(1pt) sort) (lfit close
tradeday, lwidth(2pt)) (lowess close tradeday, bwidth(.5)
lwidth(3pt))
```

If you wish, you can specify the line width in points. In this example, the width of the line for the `line` plot is 1 point, the width of the line for the `lfit` plot is 2 points, and the width of the line for the `lowess` plot is 3 points. *Uses spjanfeb2001.dta*

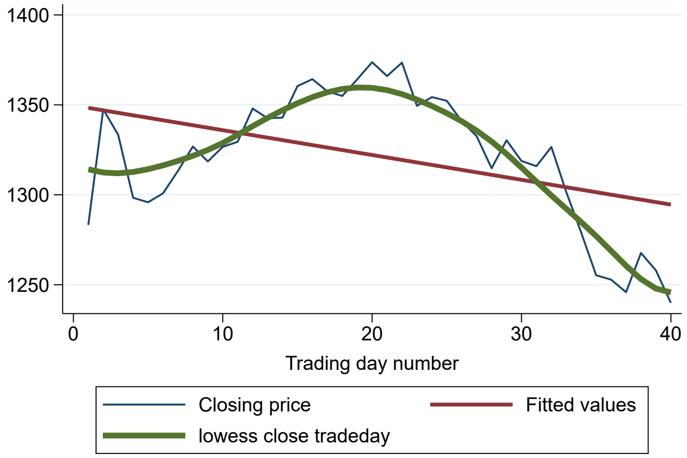


```
twoway (line close tradeday, lwidth(.01in) sort) (lfit close
tradeday, lwidth(.02in)) (lowess close tradeday, bwidth(.5)
lwidth(.03in))
```

For this example, the width of the line from the `line` command is specified as 0.01 inch, the width of the line from the `lfit` command is specified as 0.02 inch, and the width of the line created by the `lowess` command is specified as 0.03 inch. *Uses spjanfeb2001.dta*



```
twoway (line close tradeday, lwidth(.02cm) sort) (lfit close
tradeday, lwidth(.04cm)) (lowess close tradeday, bwidth(.5)
lwidth(.06cm))
```



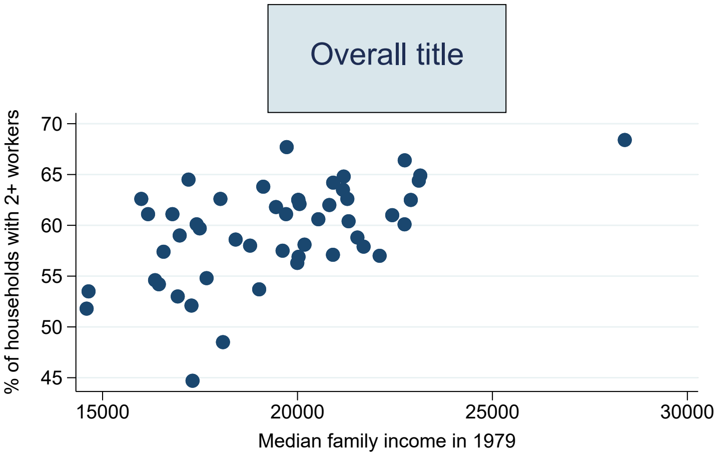
If you wish, you can specify the line widths in centimeters. In this example, I have specified the width of the line for the `line` plot as 0.02 centimeters, the width of the line for the `lfit` plot as 0.04 centimeters wide, and the width of the line showing `lowess` curve as 0.06 centimeters. *Uses spjanfeb2001.dta*

10.8 Margin

We can specify the size of a margin in a number of different ways. We can use a word that represents a predefined margin. Choices include `zero`, `vtiny`, `tiny`, `vsmall`, `small`, `medsmall`, `medium`, `medlarge`, `large`, and `vlarge`. They also include `top_bottom` to indicate a medium margin at the top and bottom and `sides` to indicate a medium margin at the left and right. A second method is to give four numbers giving the margins at the left, right, top, and bottom. A third method is to use expressions, such as `b=5` to modify one or more of the margins. These methods are illustrated below. See `help marginstyle` for more information.

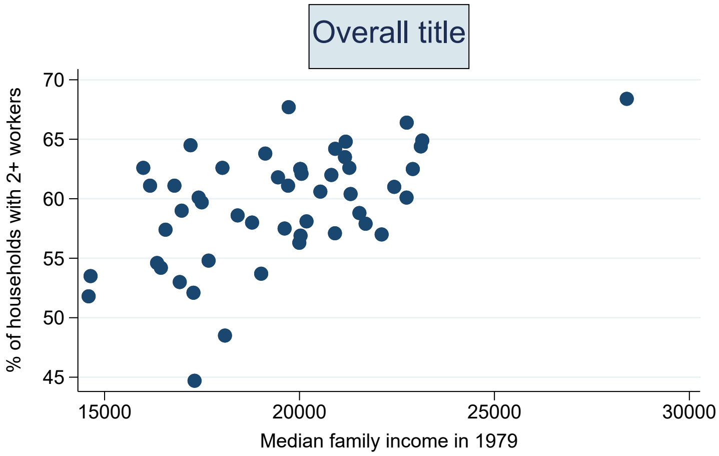
```
scatter workers2 faminc, title("Overall title", margin(large)
box)
```

We illustrate the control of margins by adding a title to this scatterplot and putting a box around it. The `margin()` option affects the distance between the title and the box. We specify a `large` margin, making the margin large on all four sides. *Uses allstatesdc.dta*



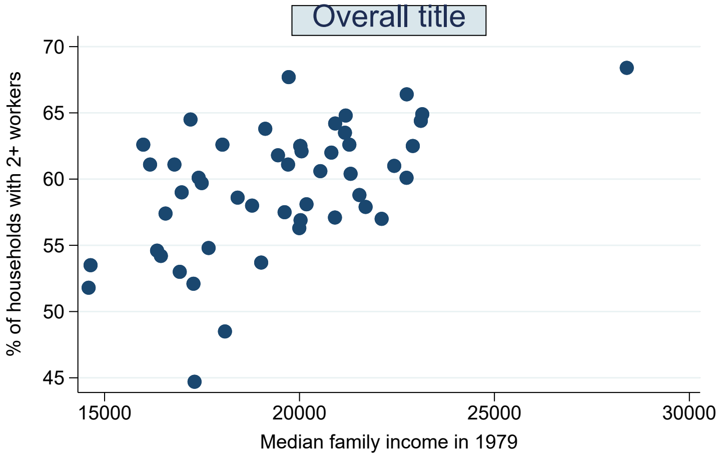
```
scatter workers2 faminc, title("Overall title",
margin(top_bottom) box)
```

By using `margin(top_bottom)`, we obtain a margin that is medium on the top and bottom but zero on the left and right. *Uses allstatesdc.dta*



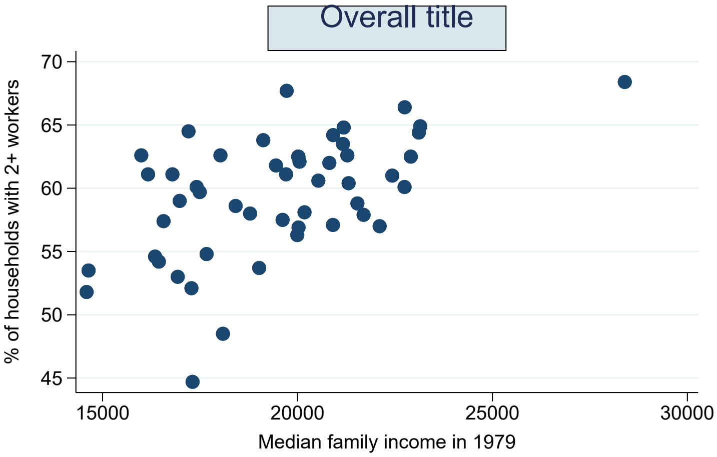
```
scatter workers2 faminc, title("Overall title", margin(sides)
box)
```

By using the `margin(sides)` option, we obtain a margin that is medium on the left and right but zero on the top and bottom. *Uses allstatesdc.dta*

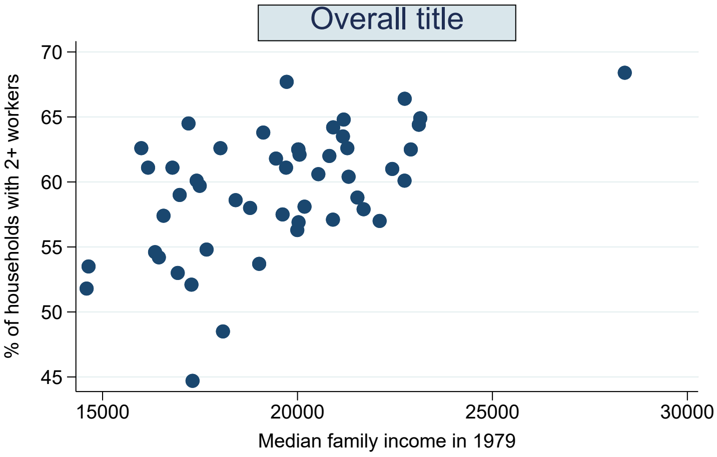


```
scatter workers2 faminc, title("Overall title", margin(10 6 3
0) box)
```

In addition to the words describing margins, we can manually specify the margins for the left, right, bottom, and top. Here we specify `margin(10 6 3 0)` and make the margin for the left 10, for the right 6, for the bottom 3, and for the top 0. *Uses allstatesdc.dta*



```
scatter workers2 faminc, title("Overall title", margin(l=10
r=10) box)
```

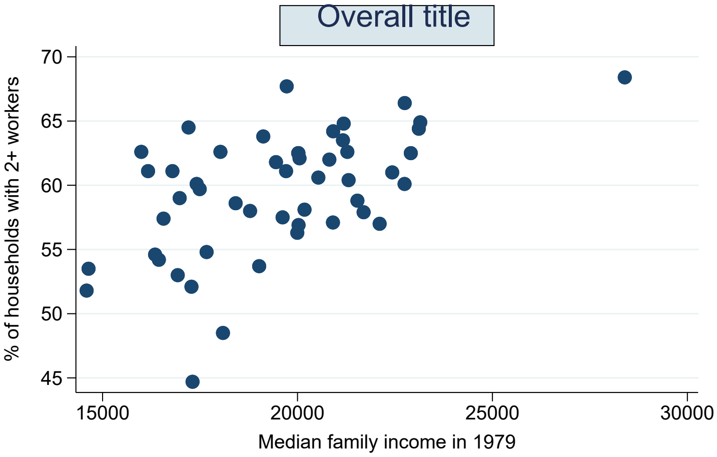


We can also manually change only some of the margins. By specifying `margin(l=10 r=10)`, we make the margins at the left and right 10 units, leaving the top and bottom unchanged. You can specify one or more of the expressions `l=`, `r=`, `t=`, or `b=` to modify the left, right, top, or bottom margins, respectively. *Uses allstatesdc.dta*

In the two examples above, I did not clearly describe what the values 10, 6, 3, and 0 referred to. They represent a percentage of the graph size (specifically, the smaller of the width or height). So, if the graph were 2 inches by 3 inches, the value of 10 would be 10% of 2 inches, or 0.2 inch. If you wish, you can specify these margins using points, inches, or centimeters. To request a margin of 10 points, you can specify `10pt`. To request a margin of half an inch, you can specify `0.5in`. To request a margin of 2 centimeters, you can specify `2cm`. If you will be resizing your graph, please see section 9.3.1, which illustrates the impact of using absolute units versus relative units when resizing graphs.

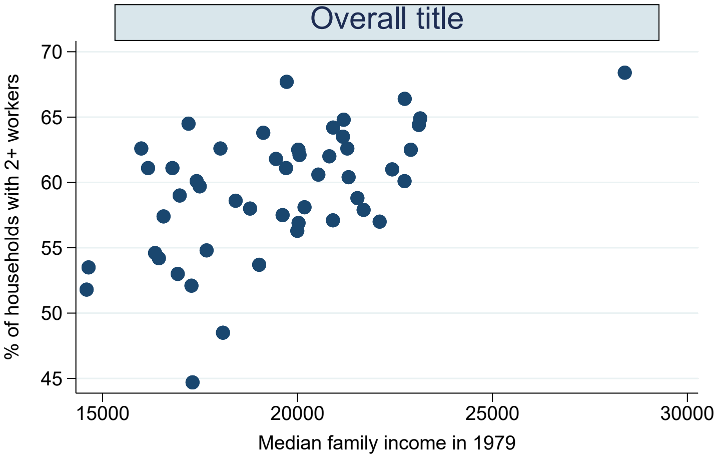
```
scatter workers2 faminc, title("Overall title", margin(10pt
6pt 3pt 0pt) box)
```

In this example, I specified `margin(10pt 6pt 3pt 0pt)`, making the margin for the left 10 points, for the right 6 points, for the bottom 3 points, and for the top 0 points. *Uses allstatesdc.dta*



```
scatter workers2 faminc, title("Overall title", margin(l=2cm
r=2cm) box)
```

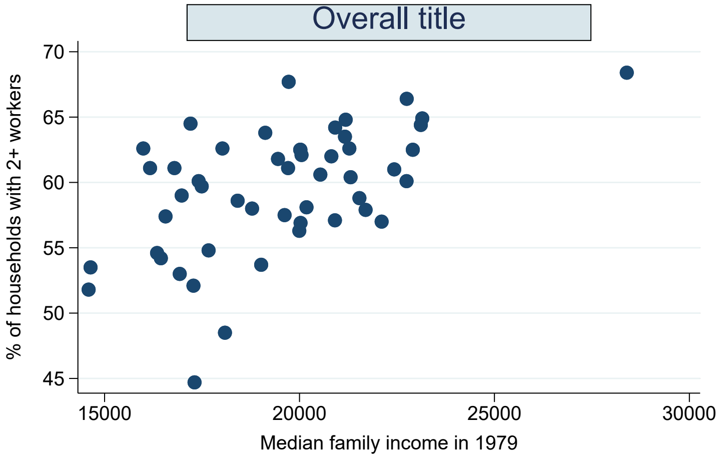
In this example, I specified `margin(l=2cm r=2cm)`, which makes the margins at the left and right each 2 centimeters wide. *Uses allstatesdc.dta*



```
scatter workers2 faminc, title("Overall title", margin(l=.5in
r=.5in) box)
```

In this example, I specified `margin(l=0.5in r=0.5in)`, which makes the margins at the left and right each half an inch wide.

Uses allstatesdc.dta

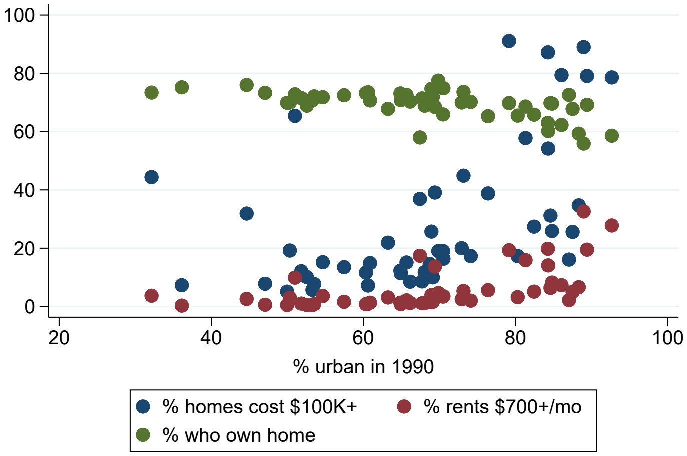


10.9 Marker size

We can specify the size of the markers in a graph using relative units or absolute units. Stata offers three methods for specifying marker sizes using relative units and three methods for specifying marker sizes using absolute units. Using relative units, you can specify a keyword that describes the size of a marker, such as `vtiny`, `tiny`, `vsmall`, `small`, `medsmall`, `medium`, `medlarge`, `large`, `vlarge`, `huge`, `vhuge`, and `ehuge`. If you prefer, you can size the marker as a multiple of the original size of the marker (for example, *2 is twice as large, or *0.7 is 0.7 times as large). Additionally, we can specify the size as a percentage of the overall size of the graph (specifically, the smaller of the height or width). If you prefer, you can specify the marker size with absolute units, using points, inches, or centimeters. (If you will be resizing your graph, please see section 9.3.1, which illustrates the impact of using absolute units versus relative units when resizing graphs.) Each of these methods is illustrated below. For more details, see `help markersizestyle` for more information.

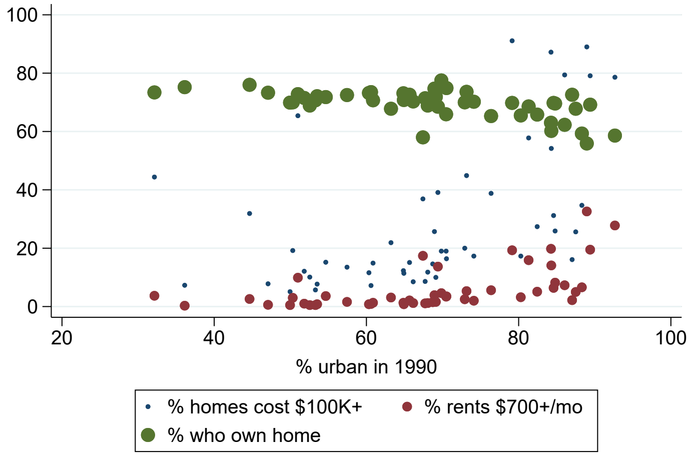
```
twoway (scatter propval100 rent700 ownhome urban)
```

Here we have an overlaid scatterplot where we graph three variables on the y axis (`propval100`, `rent700`, and `ownhome`). The markers in this graph are shown using different colors, and do not vary in size. *Uses allstatesdc.dta*



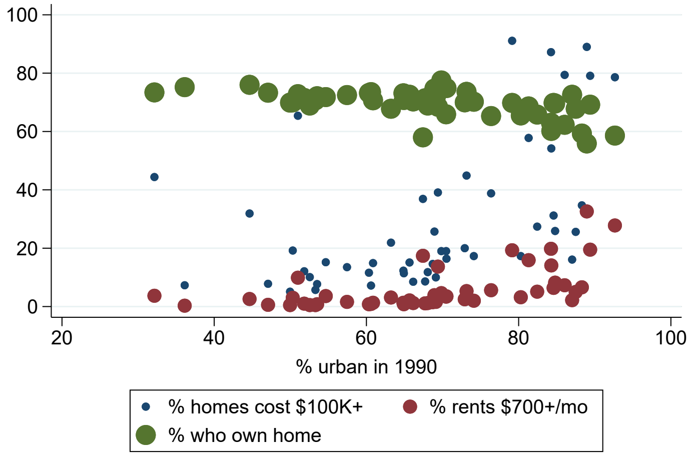
```
twoway (scatter propval100 rent700 ownhome urban,
msize(vsmall medium large))
```

I have repeated the scatterplot from the prior example, but have added the `msize(vsmall medium large)` option to make the sizes of these markers very small, medium, and large, respectively. *Uses allstatesdc.dta*



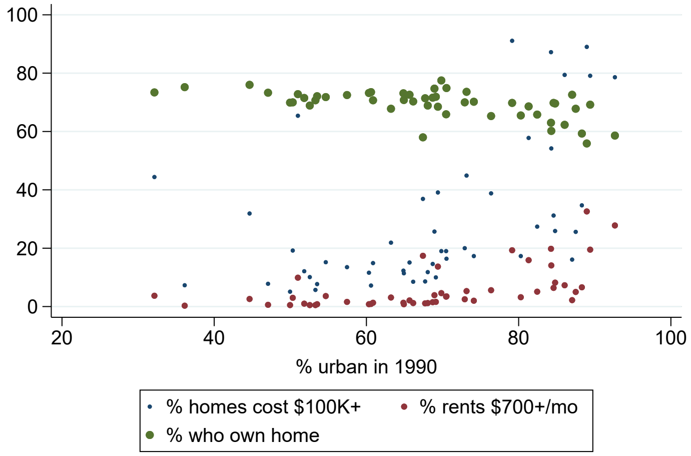
```
twoway (scatter propval100 rent700 ownhome urban, msize(*.5
*1 *1.5))
```

If you prefer, you can specify the marker size as a multiple of the original size of the marker. In this example, the first marker is displayed as half the original size, the second marker is displayed using its normal size, and the third marker is displayed as 1.5 times the original size. *Uses allstatesdc.dta*



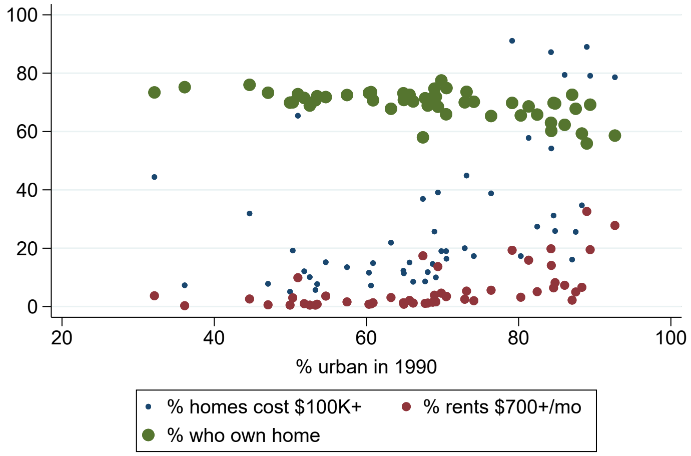
```
twoway (scatter propval100 rent700 ownhome urban, msize(0.4rs
0.8rs 1.2rs))
```

We can also specify the marker size as a percentage of the size of the graph (specifically, the smaller of the height or width). When we specify `msize(0.4rs 0.8rs 1.2rs)`, the size of the first marker is 0.4% of the size of the graph, while the second marker is 0.8% of the size of the graph, and the third marker is 1.2% of the size of the graph. *Uses allstatesdc.dta*



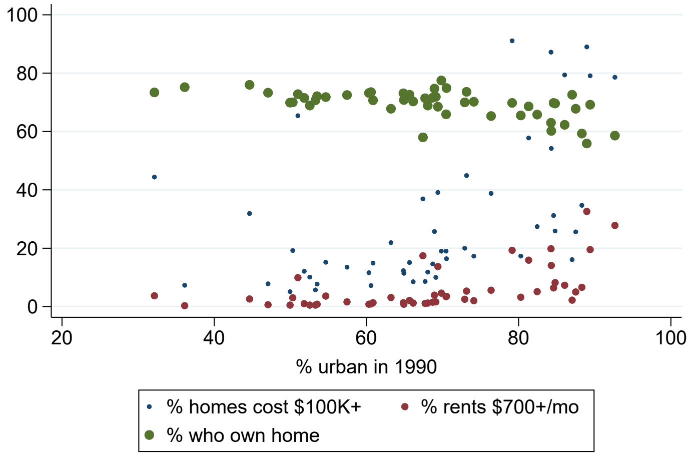
```
twoway (scatter propval100 rent700 ownhome urban, msize(1pt
2pt 3pt))
```

This example illustrates how you can control the marker sizes using printer points (that is, 1/72nd of an inch). The size of the markers in this example are 1 point, 2 points, and 3 points, respectively. *Uses allstatesdc.dta*



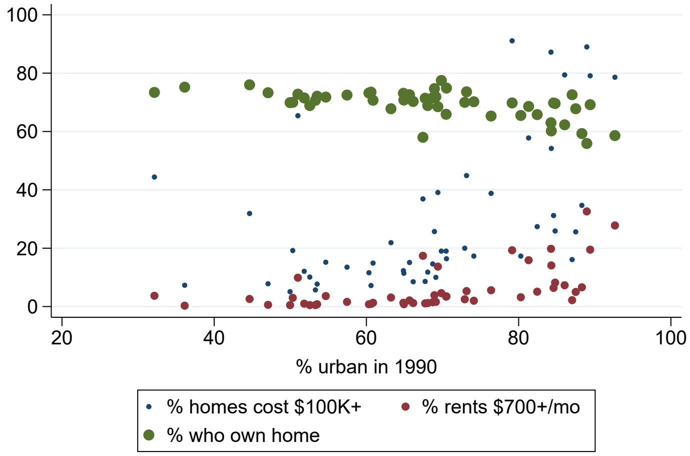
```
twoway (scatter propval100 rent700 ownhome urban, msize(.01in
.02in .03in))
```

The size of the markers can be specified in inches. In this example, the first marker is 0.01 inch, the second is 0.02 inch, and the third is 0.03 inch. *Uses allstatesdc.dta*



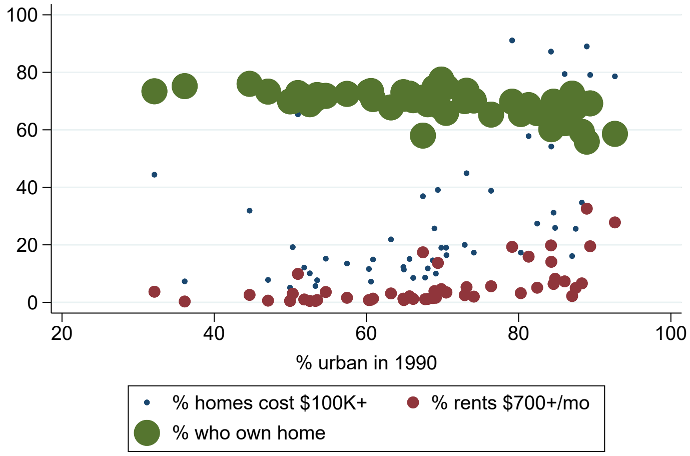
```
twoway (scatter propval100 rent700 ownhome urban, msize(.03cm
.06cm .09cm))
```

If you prefer, you can specify the size of the markers using centimeters. In this example, the first marker is 0.03 centimeters, the second is 0.06 centimeters, and the third is 0.09 centimeters. *Uses allstatesdc.dta*



```
twoway (scatter propval100 rent700 ownhome urban, msize(1pt
.1cm *2))
```

You can mix different ways of specifying the marker size within the `msize()` option. When we specify `msize(1pt .1cm *2)`) in this example, the first marker is 1 point, the second is 0.1 centimeters, and the third is twice its normal size. *Uses allstatesdc.dta*

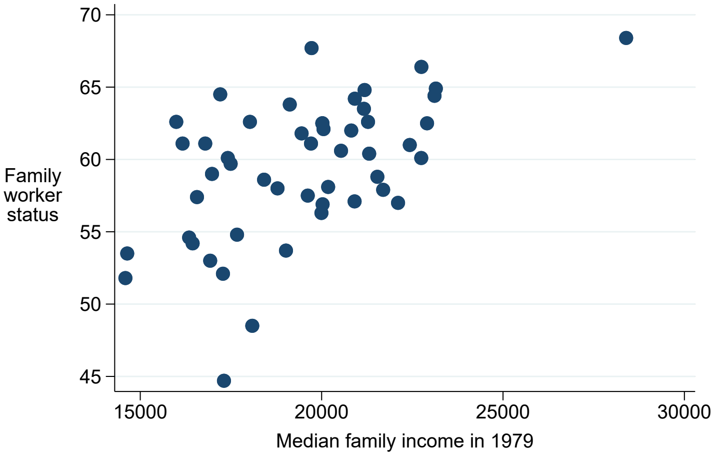

100
80
60
40
20
0
20
40
60
80
100
% urban in 1990
% homes cost $100K+
% rents $700+/mo
% who own home

10.10 Orientation

We use an *orientationstyle* to change the orientation of text, such as a y-axis title, an x-axis title, or added text. An *orientationstyle* is similar to an *anglestyle*; see Styles : Angles (section 10.1). You can specify four different orientations using the keywords `horizontal` for 0 degrees, `vertical` for 90 degrees, `rhorizontal` for 180 degrees, and `rvertical` for 270 degrees. See `help orientationstyle` for more information.

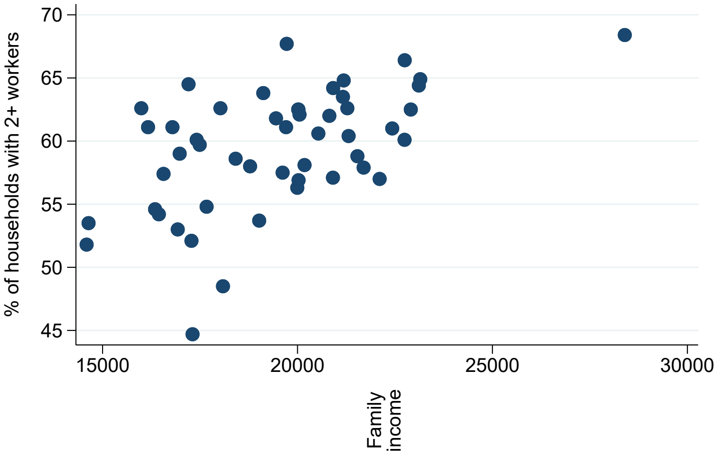
```
scatter workers2 faminc, ytitle("Family" "worker" "status",
orientation(horizontal))
```

This example shows how we can rotate the title for the y axis using the `orientation(horizontal)` option to make the title horizontal. *Uses allstatesdc.dta*



```
scatter workers2 faminc, xtitle("Family" "income",
orientation(vertical))
```

This example shows how we can rotate the title for the x axis to be vertical by using the `orientation(vertical)` option. *Uses allstatesdc.dta*

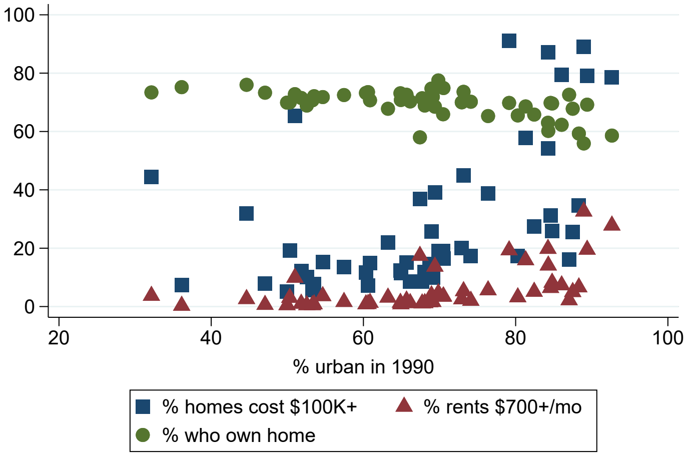

% of households with 2+ workers
70
65
60
55
50
45
15000
20000
25000
30000
Family income

10.11 Marker symbol

Stata allows us to choose among a wide variety of marker symbols via the `msymbol()` option. Let's start by considering four commonly used shapes, circles, diamonds, triangles, and squares. Specifying `msymbol(O)` produces a large circle. Or we could specify `D` to specify a diamond shape, `T` for triangles, or `S` for squares. We can also use lowercase letters `o`, `d`, `t`, and `s` to indicate smaller versions of these symbols. Further, we can append an `h` to indicate that the symbol should be displayed as hollow (for example, `Oh` requests a large hollow circle, and `oh` requests a small hollow circle). See `help symbolstyle` for more information.

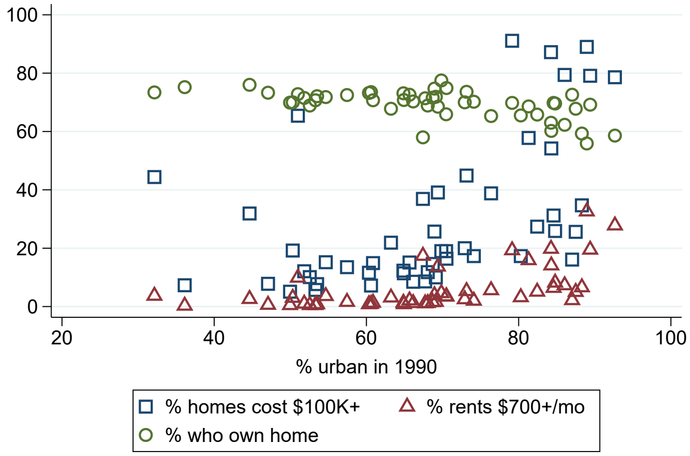
```
twoway (scatter propval100 rent700 ownhome urban, msymbol(S T O))
```

We use the `msymbol(S T O)` (marker symbol) option to plot the three symbols in this graph using squares, triangles, and circles. *Uses allstatesdc.dta*



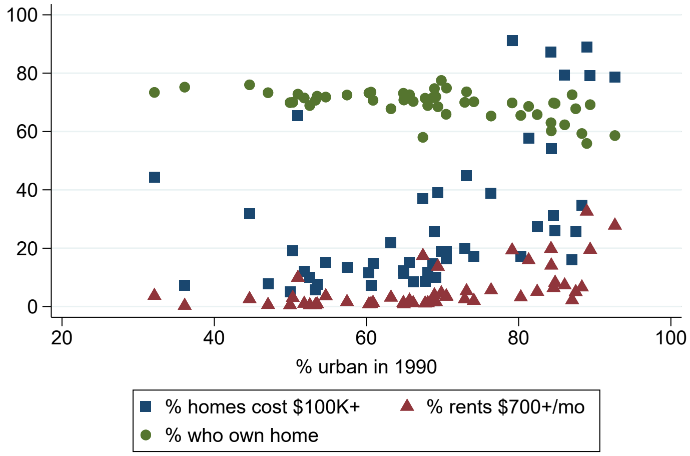
```
twoway (scatter propval100 rent700 ownhome urban, msymbol(Sh Th Oh))
```

We append an `h` to each marker symbol value to indicate that the symbol should be displayed as hollow. *Uses allstatesdc.dta*



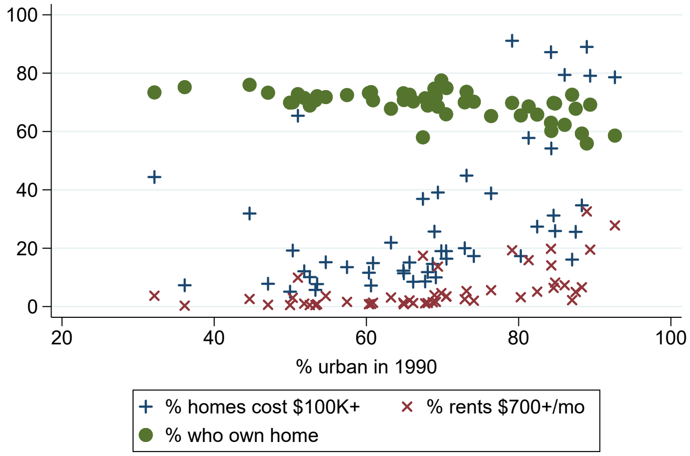
```
twoway (scatter propval100 rent700 ownhome urban, msymbol(s t
o))
```

Here we use the `msymbol(s t o)` option to specify small squares, small triangles, and small circles. *Uses allstatesdc.dta*

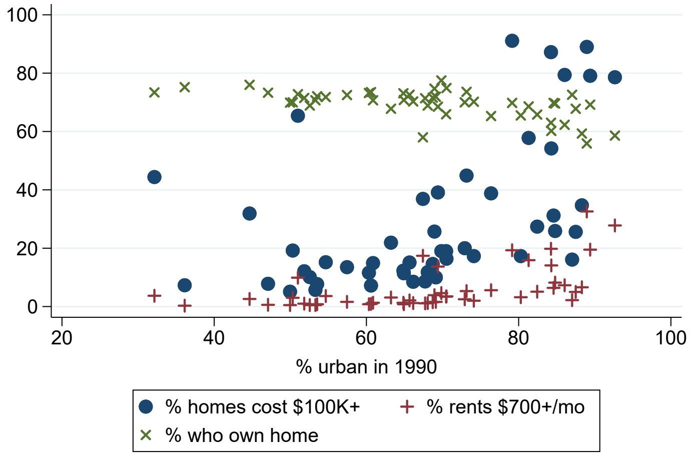


```
twoway (scatter propval100 rent700 ownhome urban, msymbol(+
X))
```

This graph introduces two more symbols, a large plus sign and a large X. When we specify the `msymbol(+ X)` option, the first variable (that is, `propval100`) is displayed using a large plus sign, and the second variable (that is, `rent700`) is displayed using a large X. The third variable (that is, `ownhome`) is displayed using the default symbol (a circle). *Uses allstatesdc.dta*



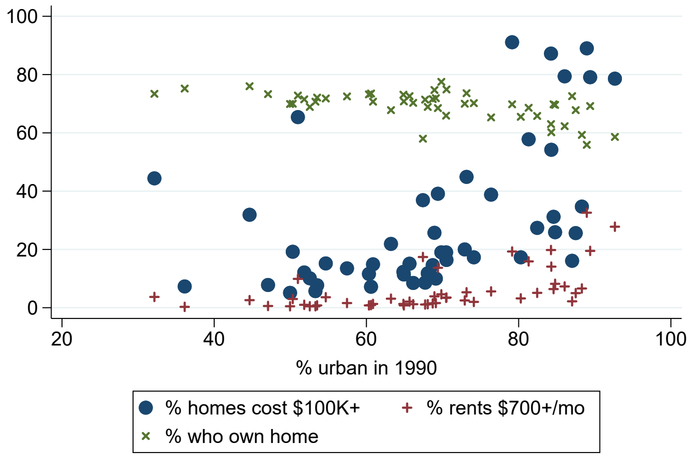
```
twoway (scatter propval100 rent700 ownhome urban, msymbol(. +
X))
```



In the graph above, it is hard to distinguish the plus sign from an X, in part because points of the scatterplot from the first two variables show considerable overlap. This example uses the `msymbol(. + X)` option to display the first variable using the default symbol and the second and third variables using a large plus sign and a large X (respectively). *Uses allstatesdc.dta*

```
twoway (scatter propval100 rent700 ownhome urban, msymbol(.
smplus x))
```

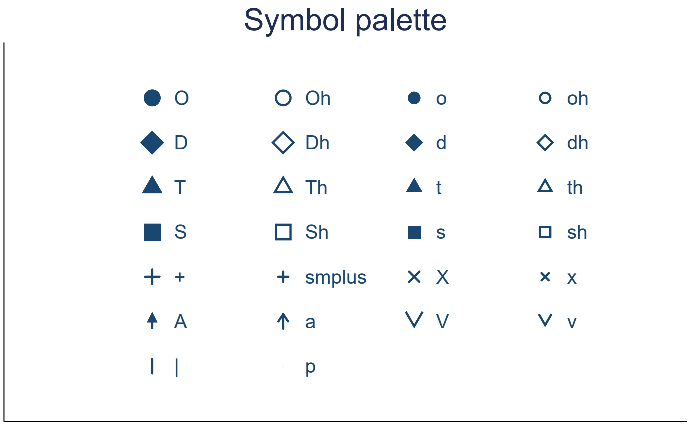
This graph is like the one above, but instead I have specified the second variable should be displayed using a small plus sign and the third variable should be displayed using a small x. *Uses allstatesdc.dta*



The graph below shows the graph created by the `palette symbolpalette` command, which shows all symbols that Stata can display. We have already seen the symbols shown in the first five rows (that is, the circles, diamonds, triangles, squares, plus signs, and Xs). The sixth row shows four directional symbols. You can obtain a large arrowhead by specifying `A`, a small arrowhead by specifying `a`, a large down-pointing caret by specifying `V`, and a small down-pointing caret by specifying `v`. The last row shows that you can display a vertical bar by specifying `|` and you can display a small fine point by specifying `p`. You can control the angle of the display of the symbols with the `msangle()` option. I think you are most likely to use this option in conjunction with the arrows, carets, and possibly the pipe.

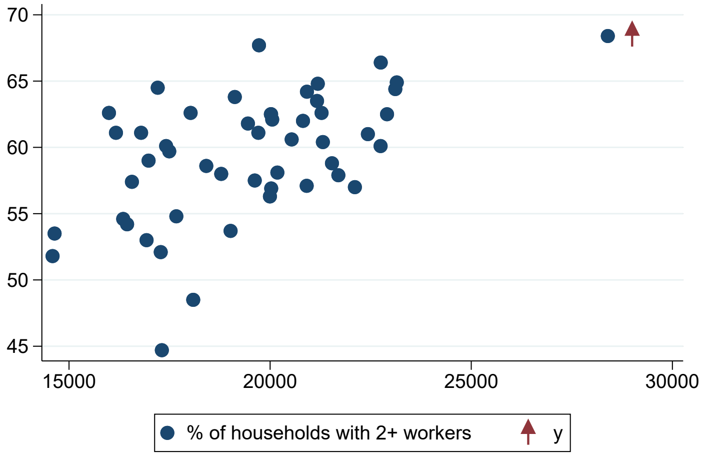
```
palette symbolpalette
```

The `palette symbolpalette` command displays all marker symbols that you can specify on the `msymbol()` option. As noted in the graph, the symbols are displayed larger than their default size to accentuate the detail in the display of the markers. *Uses allstatesdc.dta*



(symbols shown at larger than default size)

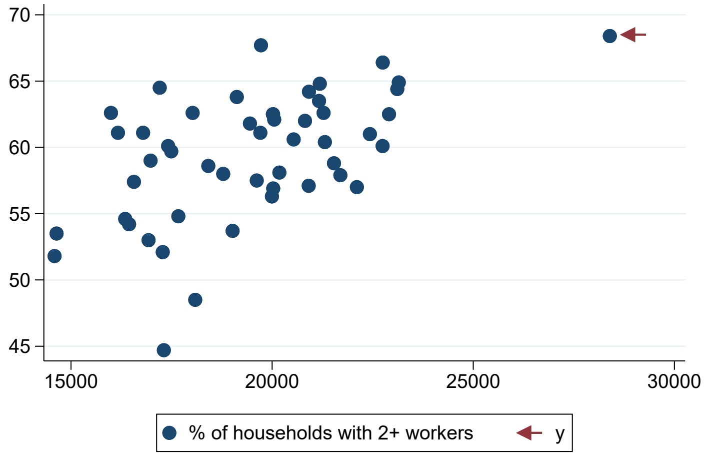
```
twoway (scatter workers2 faminc) (scatteri 68.5 29000,
msymbol(A) msize(huge))
```



Consider this scatterplot with a potential outlier. To call attention to this observation, I use the `scatteri` command to display a marker adjacent to the observation. When we specify the `msymbol(A)` option, this marker is drawn as a large filled arrow head. Note how the arrow points up. That is the default orientation. The next example illustrates use of the `msangle()` option to rotate the arrow. *Uses allstatesdc.dta*

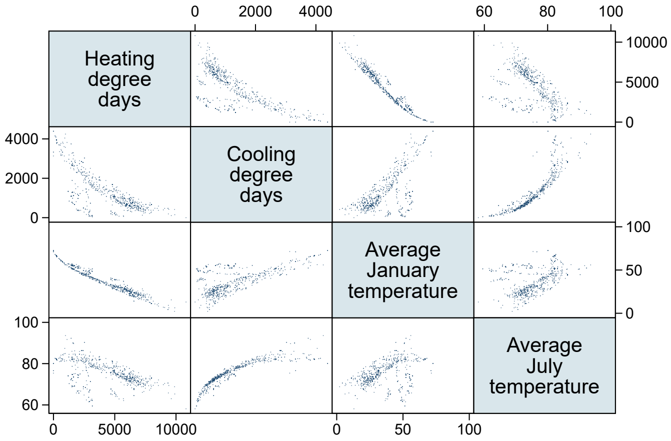
```
twoway (scatter workers2 faminc) (scatteri 68.5 29000,
msymbol(A) msangle(90) msize(huge))
```

This example adds the `msangle(90)` option. This rotates the arrow 90 degrees counterclockwise. Thus, the arrow now points to the left, pointing to the potential outlier. If we wanted the arrow to point down, we can specify `msangle(180)`, and to make the arrow point to the right, we can specify `msangle(270)`. *Uses allstatesdc.dta*



```
graph matrix heatdd cooldd tempjan tempjuly, msymbol(p)
```

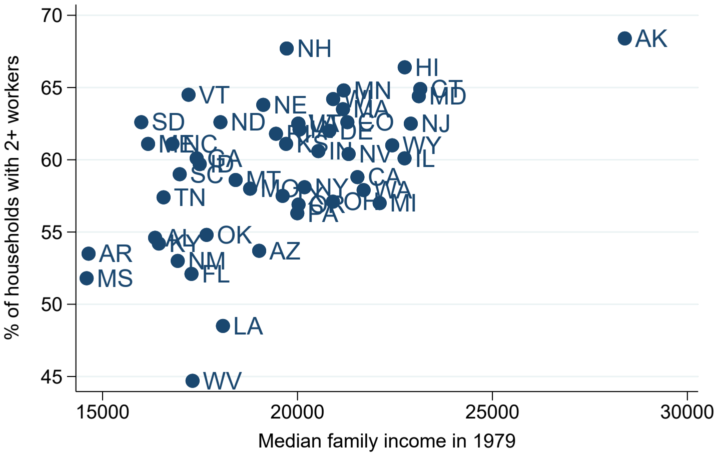
When there are many observations, the `msymbol(p)` option can be useful because it displays a small point for each observation and can help us to see the overall relationships among the variables. Here we switch to the `citytemp` data file to illustrate this option. *Uses citytemp.dta*



10.12 Text size

We can control the size of text in a graph by specifying the desired text size via absolute units or relative units. Stata offers three methods for specifying text size using absolute units: by specifying the size using points, inches, or centimeters. You can also specify sizes using relative units. The first method is the use of keywords to specify the text size—the keywords include `zero`, `minuscule`, `quarter_tiny`, `third_tiny`, `half_tiny`, `tiny`, `vsmall`, `small`, `medsmall`, `medium`, `medlarge`, `large`, `vlarge`, `huge`, and `vhuge`. The second method is specifying a multiple of the original text size (for example, *2 is twice as large, or *0.7 is 0.7 times as large). The third method specifies the size of text as a percentage of the overall size of the graph (specifically, the smaller of the height or width). Each of these methods is illustrated below. For more details, see `help textsizestyle`. (If you will be resizing your graph, please see section 9.3.1, which illustrates the impact of using absolute units versus relative units when resizing graphs.)

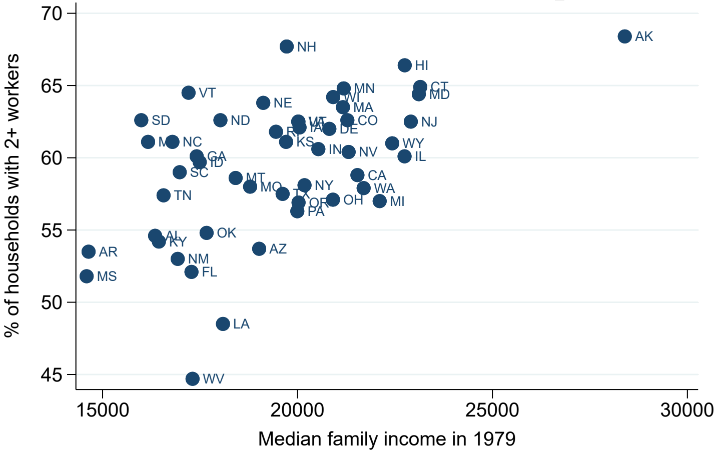
```
scatter workers2 faminc, mlabel(stateab)
```



This example uses `mlabel(stateab)` to add marker labels with the state abbreviation labeling each point. The following examples will illustrate ways to control the size of the text associated with the marker label via the `mlabsize()` option. You can use the same strategies to control the size of other text elements in a Stata graph. *Uses allstatesdc.dta*

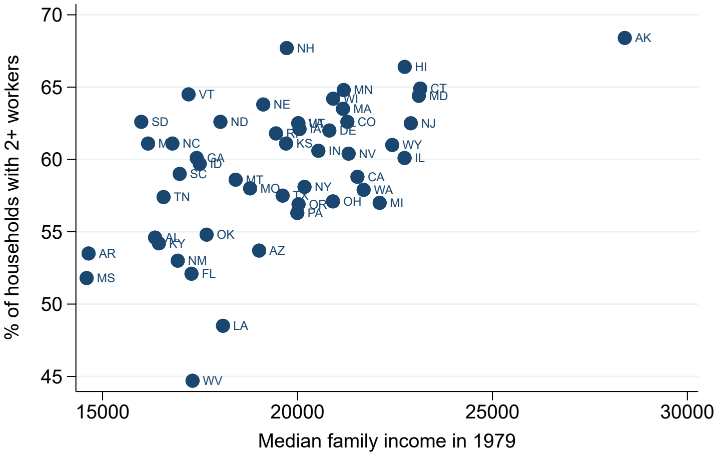
```
scatter workers2 faminc, mlabel(stateab) mlabsize(small)
```

We can control the size of the marker labels via the `mlabsize()` option. In this example, I specified `mlabsize(small)` to make the marker labels small. *Uses allstatesdc.dta*



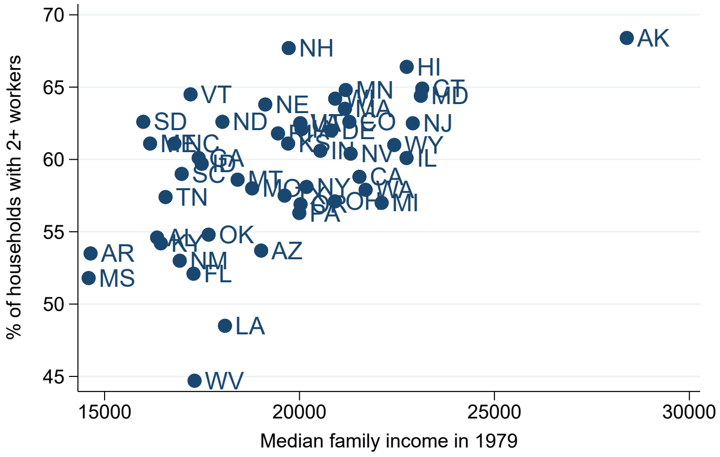
```
scatter workers2 faminc, mlabel(stateab) mlabsize(*.5)
```

Instead of using the keywords, we can specify the size as a multiple of the original size (for example, *2 is twice as big, or *0.7 is 0.7 times as big). Here we use the `mlabsize(*.5)` option to make the marker labels half as large as they would normally be. *Uses allstatesdc.dta*



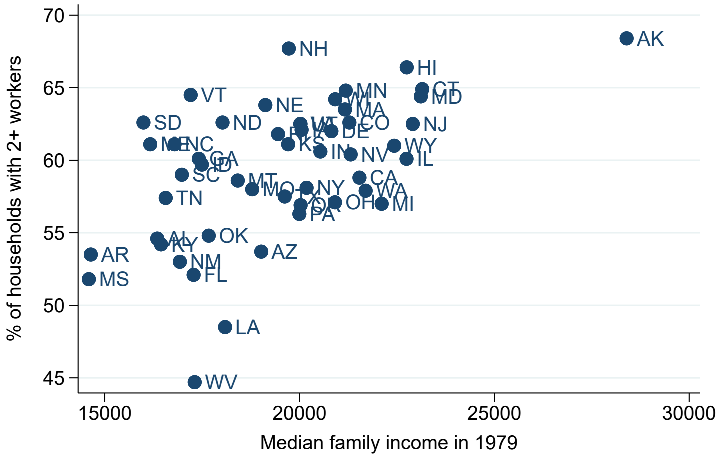
```
scatter workers2 faminc, mlabel(stateab) mlabsize(5rs)
```

We can specify the size of the marker label using a *relative size* specification, sizing the marker label as a percentage of the graph size (specifically, the smaller of the height or width). This graph is 3 inches by 2 inches, so the smaller of the sizes is 2 inches. When we specify `mlabsize(5rs)`, the markers are sized as 5% of 2 inches, or 0.1 inch. *Uses allstatesdc.dta*



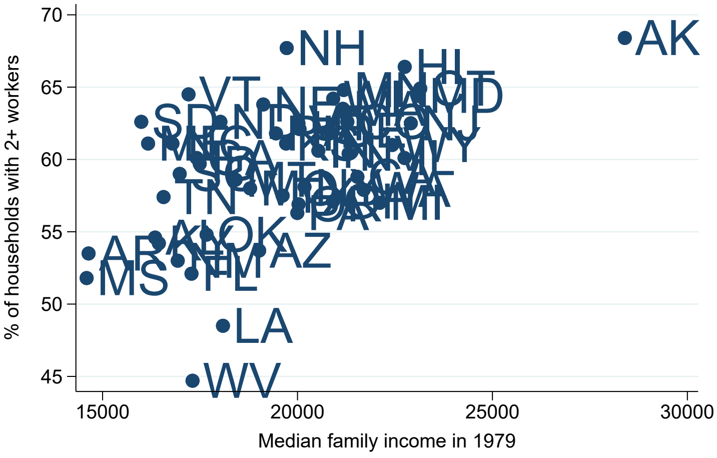
```
scatter workers2 faminc, mlabel(stateab) mlabsize(6pt)
```

We can use absolute units to specify the size of text on a graph. In this example, I specified `mlabsize(6pt)` to specify the size of the marker labels as 6 points (that is, 6/72nds of an inch). *Uses allstatesdc.dta*



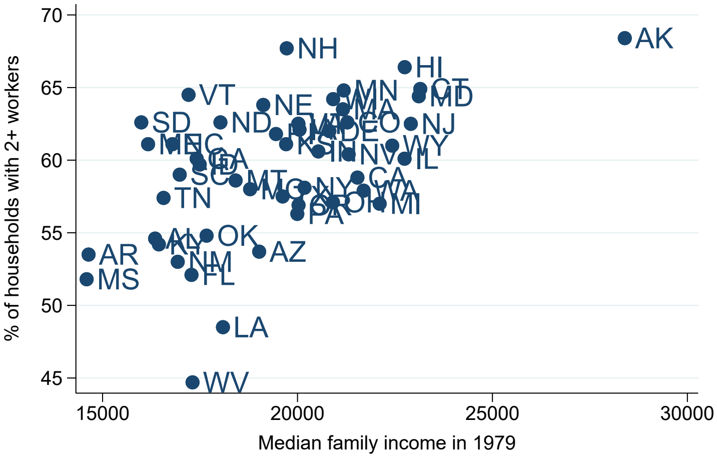
```
scatter workers2 faminc, mlabel(stateab) mlabsize(0.2in)
```

Instead of specifying the size in printer points, we can specify the size of text using inches. In this example, I specified `mlabsize(0.2in)` to specify the size of the marker labels as 0.2 inch. *Uses allstatesdc.dta*



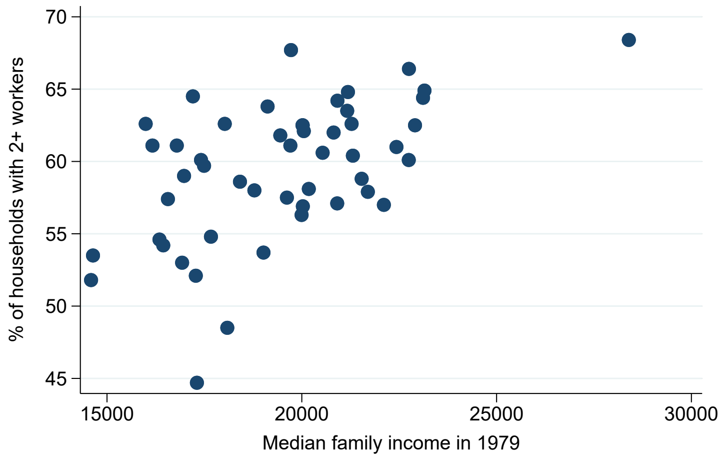
```
scatter workers2 faminc, mlabel(stateab) mlabsize(0.3cm)
```

You can also specify the size of text in centimeters. In this example, I used the option `mlabsize(0.3cm)` to display the marker label as 0.3 centimeters. *Uses allstatesdc.dta*

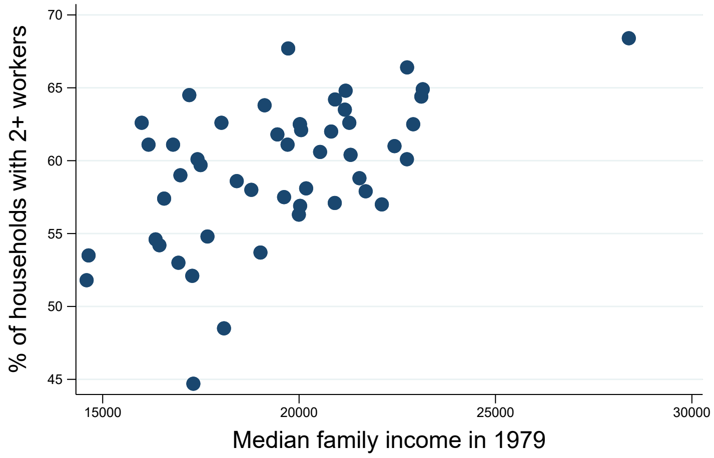


```
scatter workers2 faminc
```

Consider this graph. Note the size of the axis titles. Note the size of the axis labels. Imagine that we want to publish this figure but we have been asked to make the axis titles larger and the axis labels smaller. *Uses allstatesdc.dta*



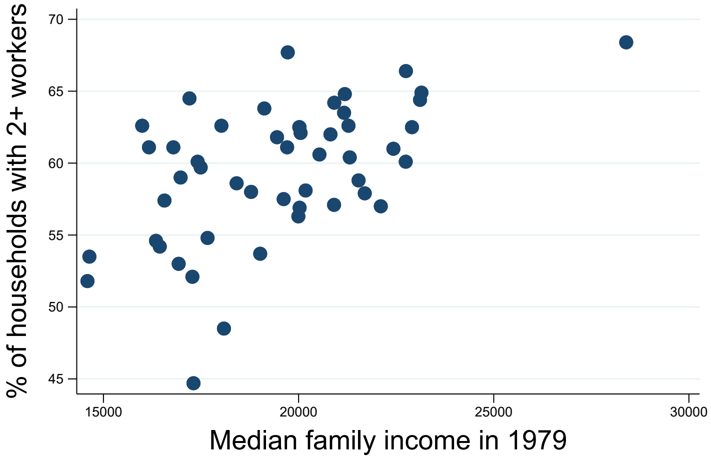
```
scatter workers2 faminc, xtitle( , size(large)) ytitle( ,
size(large)) xlabel( , labsize(small)) ylabel( ,
labsize(small))
```



In this example, I have used the `size(large)` suboption to make the size of the title `large` for the x and y axes. Also, I used the `labsize(small)` suboption to make the axis labels small for both the x and y axes. *Uses allstatesdc.dta*

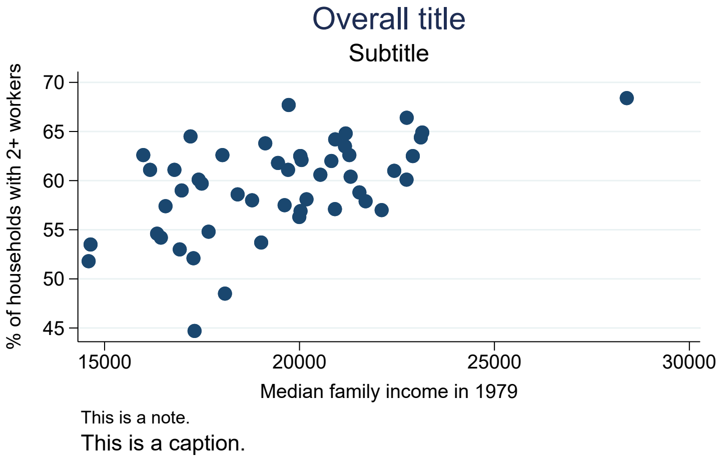
```
scatter workers2 faminc, xtitle( , size(8pt)) ytitle( ,
size(8pt)) xlabel( , labsize(4pt)) ylabel( , labsize(4pt))
```

In this example, I have used the `size(8pt)` suboption to specify the size of the title on the x and y axes as 8 points. Also, I used the `labsize(4pt)` suboption to specify that the size of the x-axis and y-axis labels as 4 points. *Uses allstatesdc.dta*



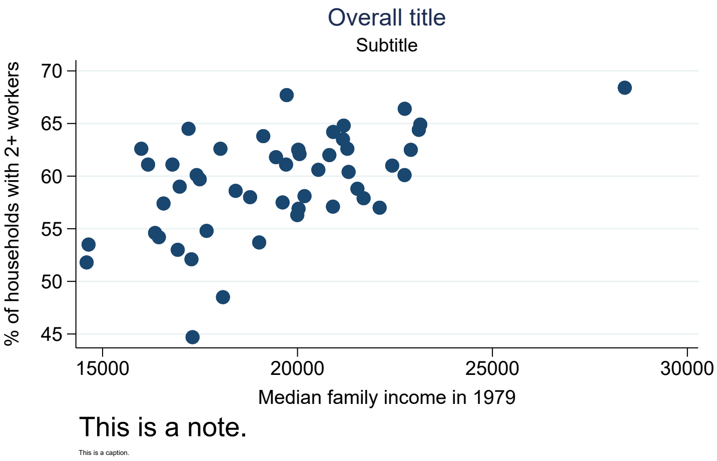
```
scatter workers2 faminc, title("Overall title")
subtitle("Subtitle") note("This is a note.") caption("This is
a caption.")
```

In this example, I have used options to add a title, subtitle, note, and caption to the graph. In the next example, I will use the `size()` option to modify the text size of each of these. *Uses allstatesdc.dta*



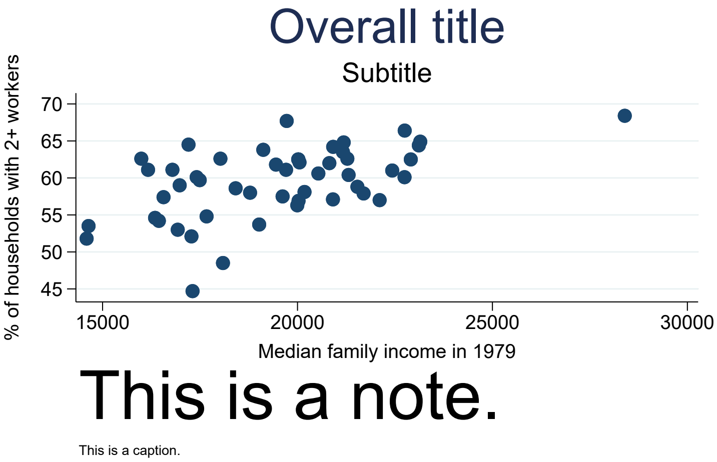
```
scatter workers2 faminc, title("Overall title", size(large))
subtitle("Subtitle", size(medium)) note("This is a note.",
size(vlarge)) caption("This is a caption.", size(tiny))
```

Note how the `size()` option is used to change the size of the display of the title, subtitle, caption, and note. I added the `size(large)` suboption to make the title large. Likewise, I made the subtitle `medium` in size, the note `vlarge`, and the caption `tiny`. *Uses allstatesdc.dta*



```
scatter workers2 faminc, title("Overall title", size(14pt))
subtitle("Subtitle", size(8pt)) note("This is a note.",
size(20pt)) caption("This is a caption.", size(4pt))
```

I have repeated the example from above but now use points to change the size of the text items. I added the `size(14pt)` suboption to display the title using 14-point type. Similarly, I have specified the size of the subtitle as 8 point, the note as 20 points, and the caption as 4 points. Also, see section 9.3.1 about resizing graphs with elements using absolute units. *Uses allstatesdc.dta*



Chapter 11
Appendix

The appendix contains a mixture of material that did not fit well in any previous chapter. The appendix shows other kinds of statistical graphs Stata can produce that were not covered in the chapters and shows how to use the options illustrated in this book to make them. Next the `marginsplot` command is illustrated, focusing on how you can use options to customize such graphs. The next section looks at how to save graphs, redisplay graphs, and combine multiple graphs into one. This is followed by a section with more realistic examples that require a combination of multiple options or data manipulation to create the graph. The appendix reviews some common mistakes in writing graph commands and shows how to fix them, followed by a brief look at creating custom schemes.

11.1 Overview of statistical graph commands

This section illustrates some of the Stata commands for producing specialized statistical graphs. Unlike other sections of this book, this section merely illustrates these kinds of graphs but does not further explain the syntax of the commands used to create them. The graphs are illustrated on the following six pages, with multiple graphs on each page. The title of each graph is the name of the Stata command that produced the graph. You can use the `help` command to find out more about that command, or you can find more information in the appropriate Stata manual. The figures are described below.

- Figure 11.1 illustrates several graphs used to examine the univariate distribution of variables.
- Figure 11.2 illustrates the `gladder` and `qladder` commands, which show the distribution of a variable according to the *ladder of powers* to help visually identify transformations for achieving normality.
- Figure 11.3 shows several graphs you can use to assess how your data meet the assumptions of linear regression.
- Figure 11.4 shows some plots that help to illustrate the results of a survival analysis.
- Figure 11.5 shows several different plots used to understand the nature of time-series data and to select among different time-series models.
- Figure 11.6 shows plots associated with receiver operating characteristic (ROC) analyses, which you can also use with logistic regression analysis.
- Figure 11.7 shows a forest plot, a graphical technique used in meta-analysis, which is created by the `meta forestplot` command.
- Figure 11.8 shows graphs of item information functions and test information functions generated by the `irtgraph` command after two different item response theory (IRT) models.
- Figure 11.9 displays graphs of impulse–response functions (IRFs) and forecast-error variance decompositions (FEVDs) after a vector autoregressive (VAR) model.

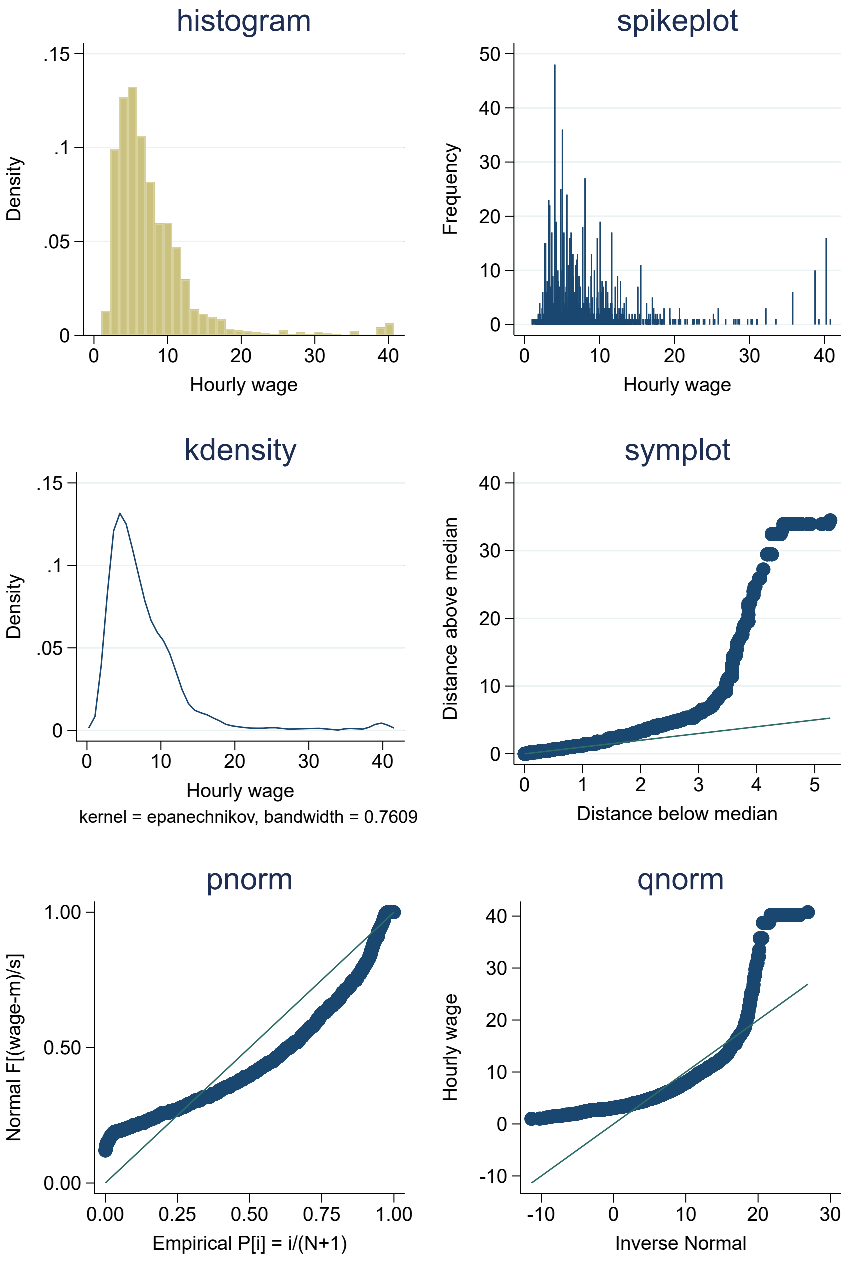
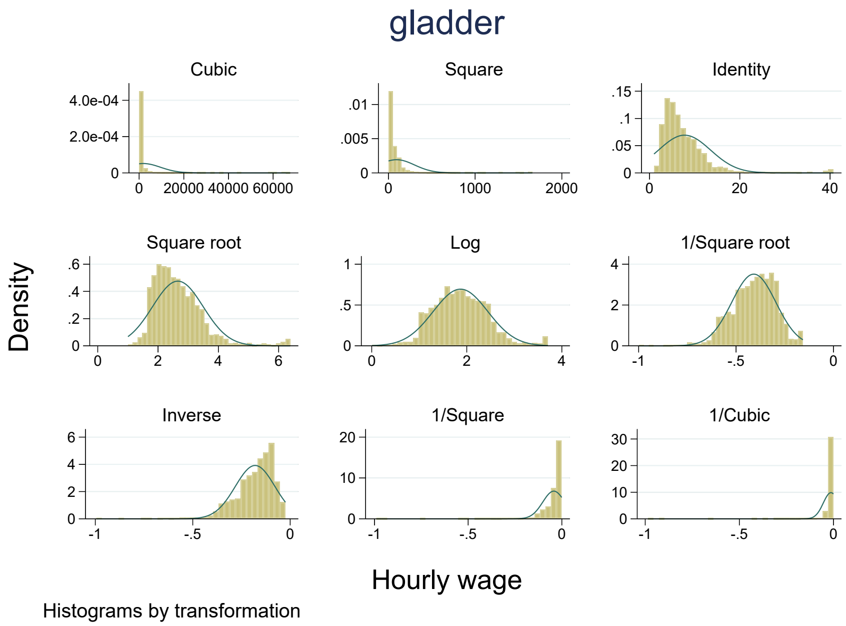


Figure 11.1. Distribution graphs



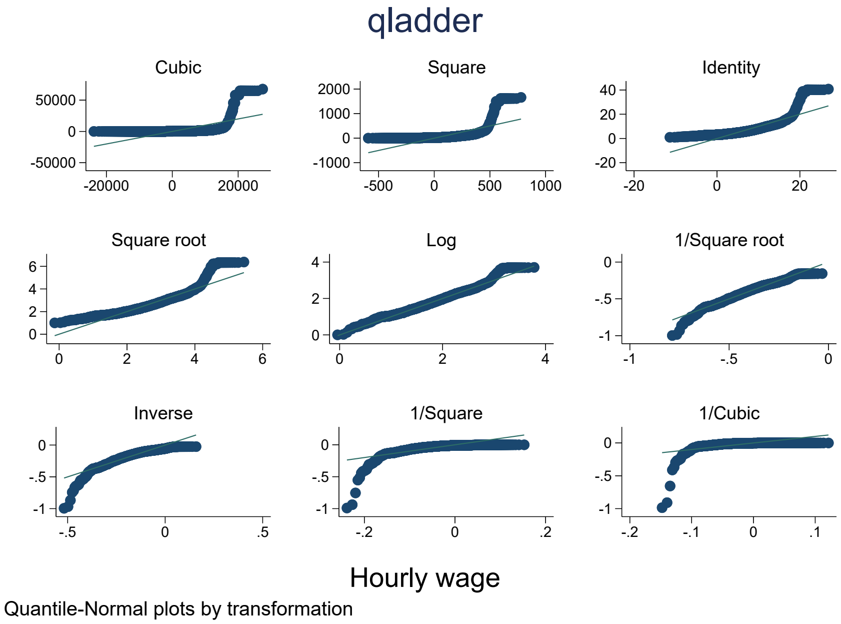


Figure 11.2. Ladder of powers graphs

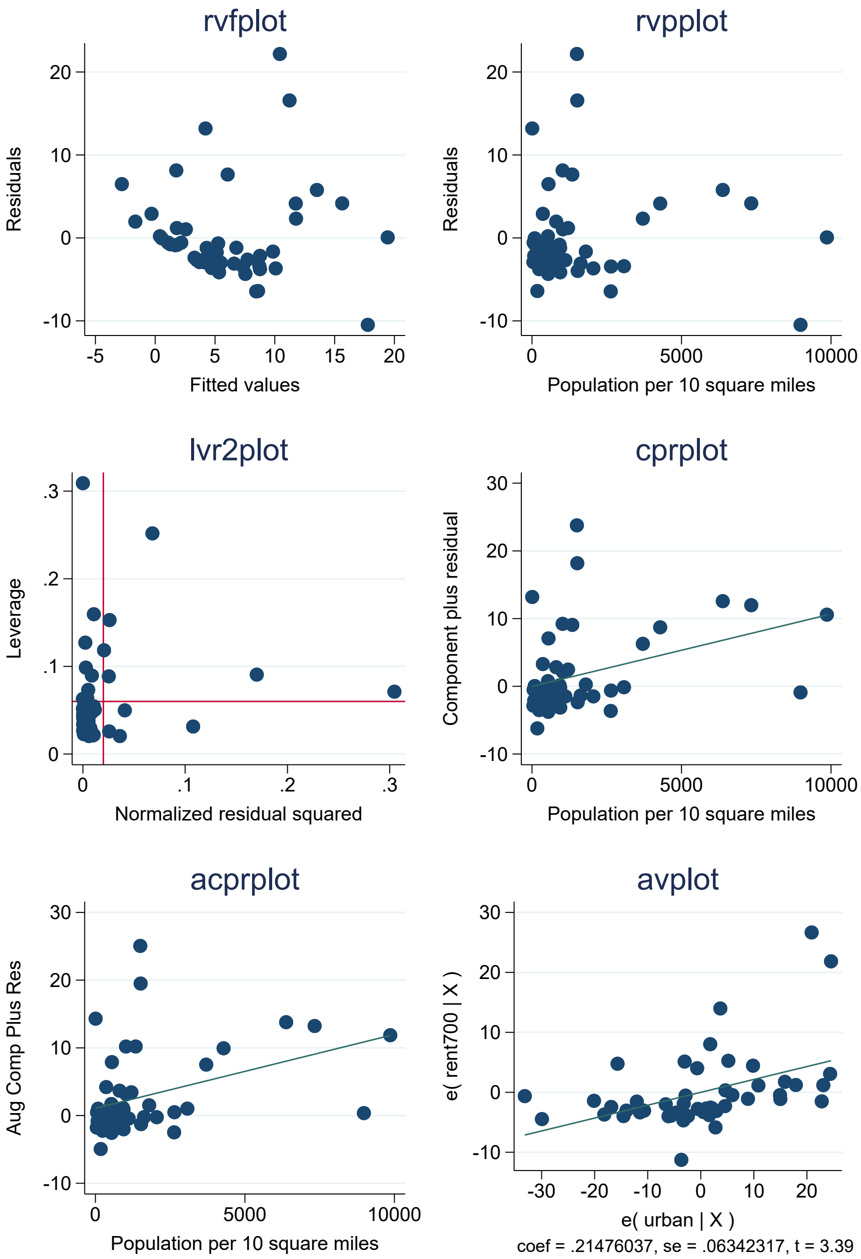
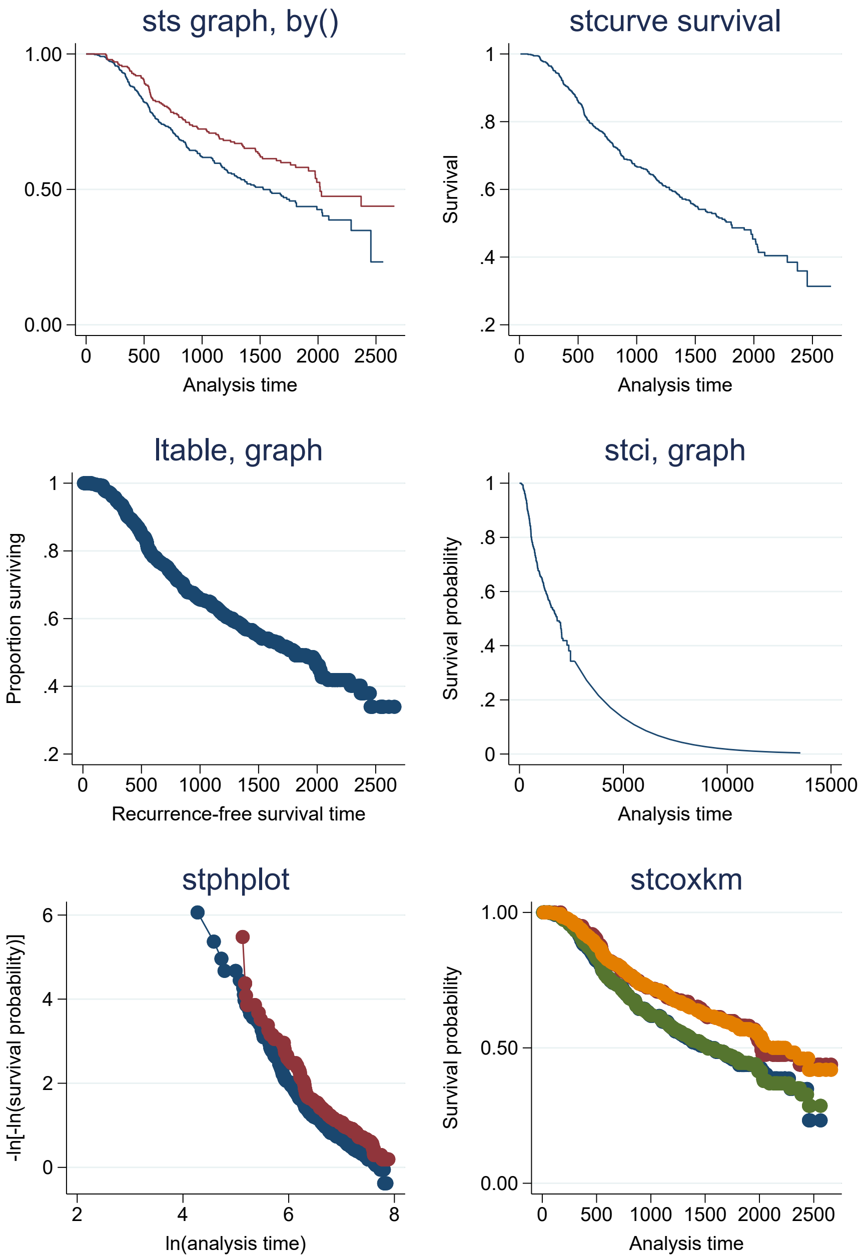


Figure 11.3. Regression diagnostics graphs


sts graph, by()

stcurve survival

ltable, graph

stci, graph

stphplot

stcoxkm

Figure 11.4. Survival graphs

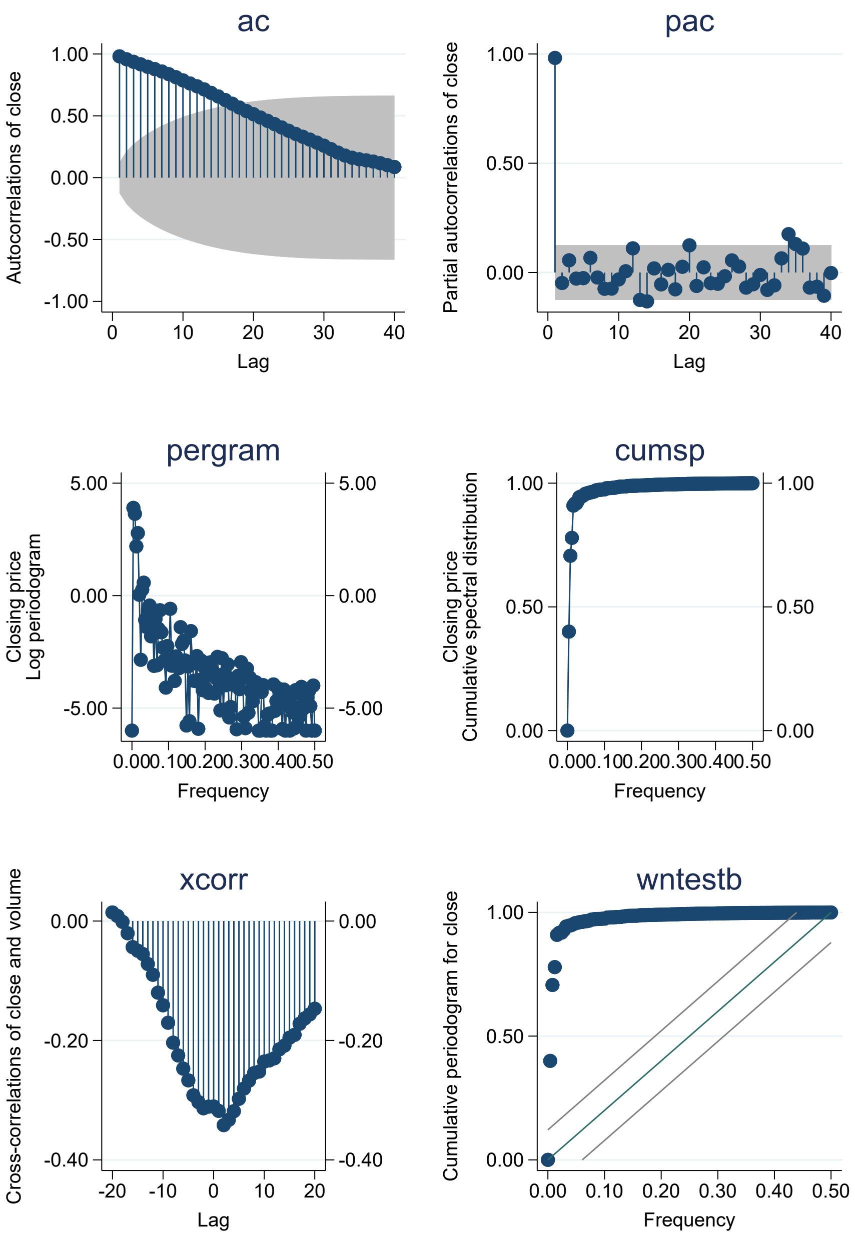


Figure 11.5. Time-series graphs

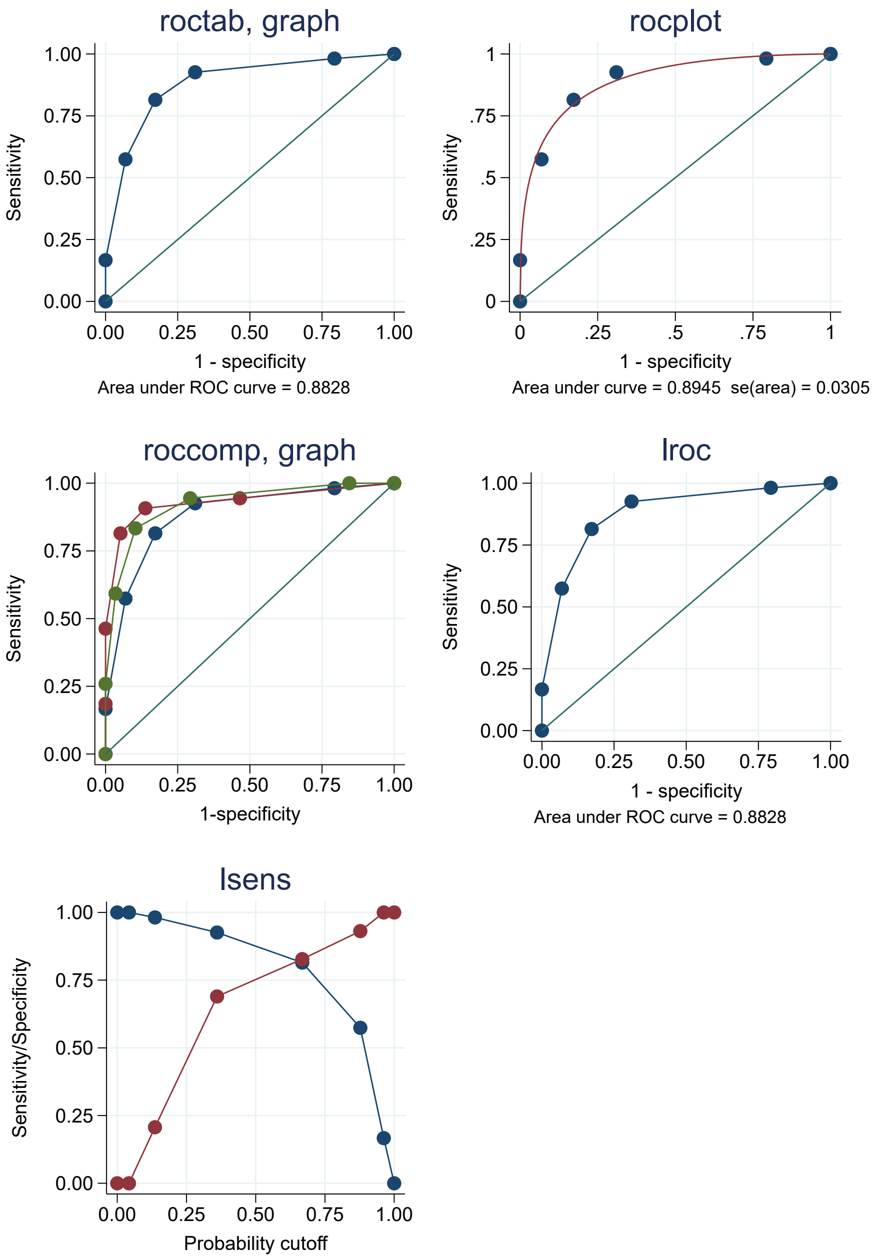


Figure 11.6. ROC graphs

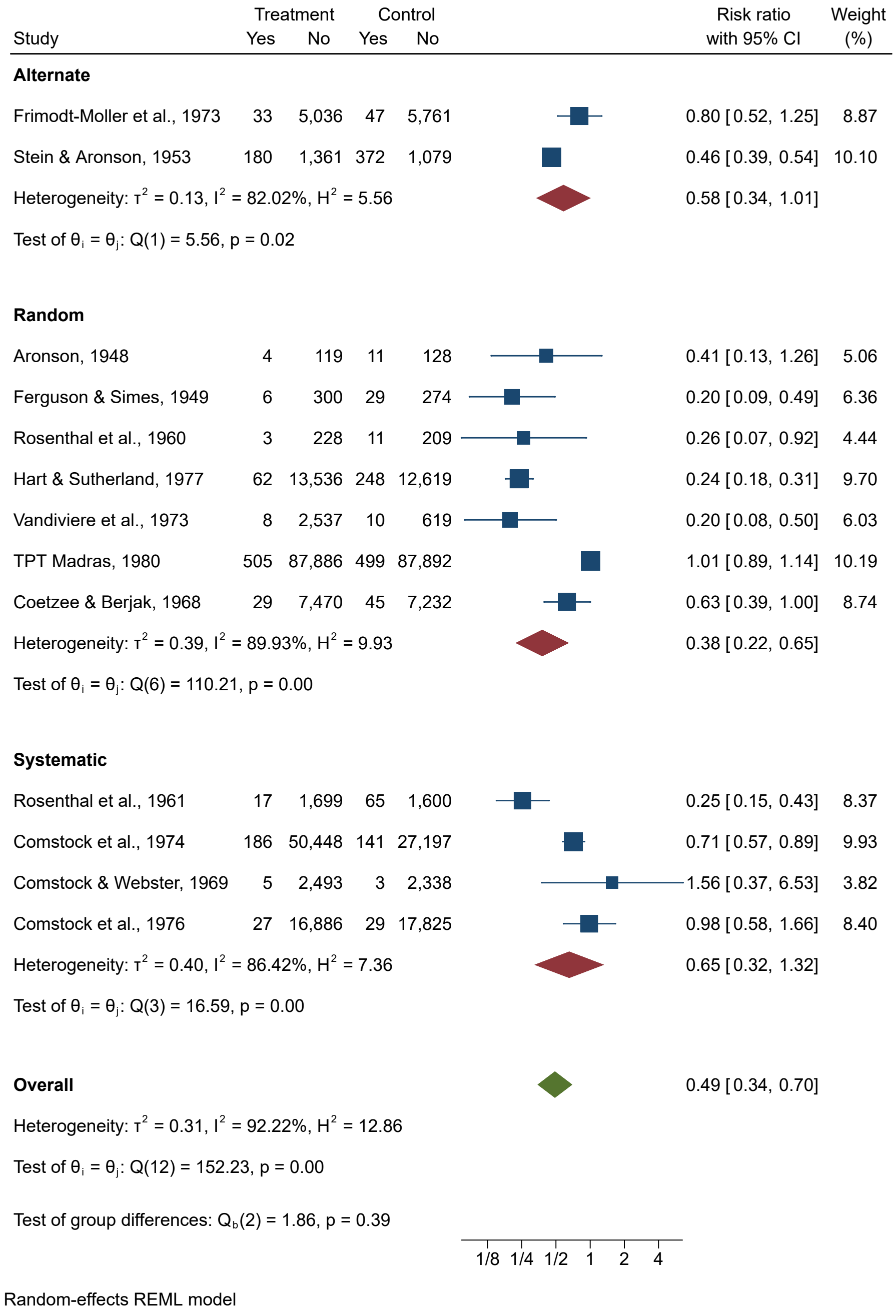


Figure 11.7. Forest plot

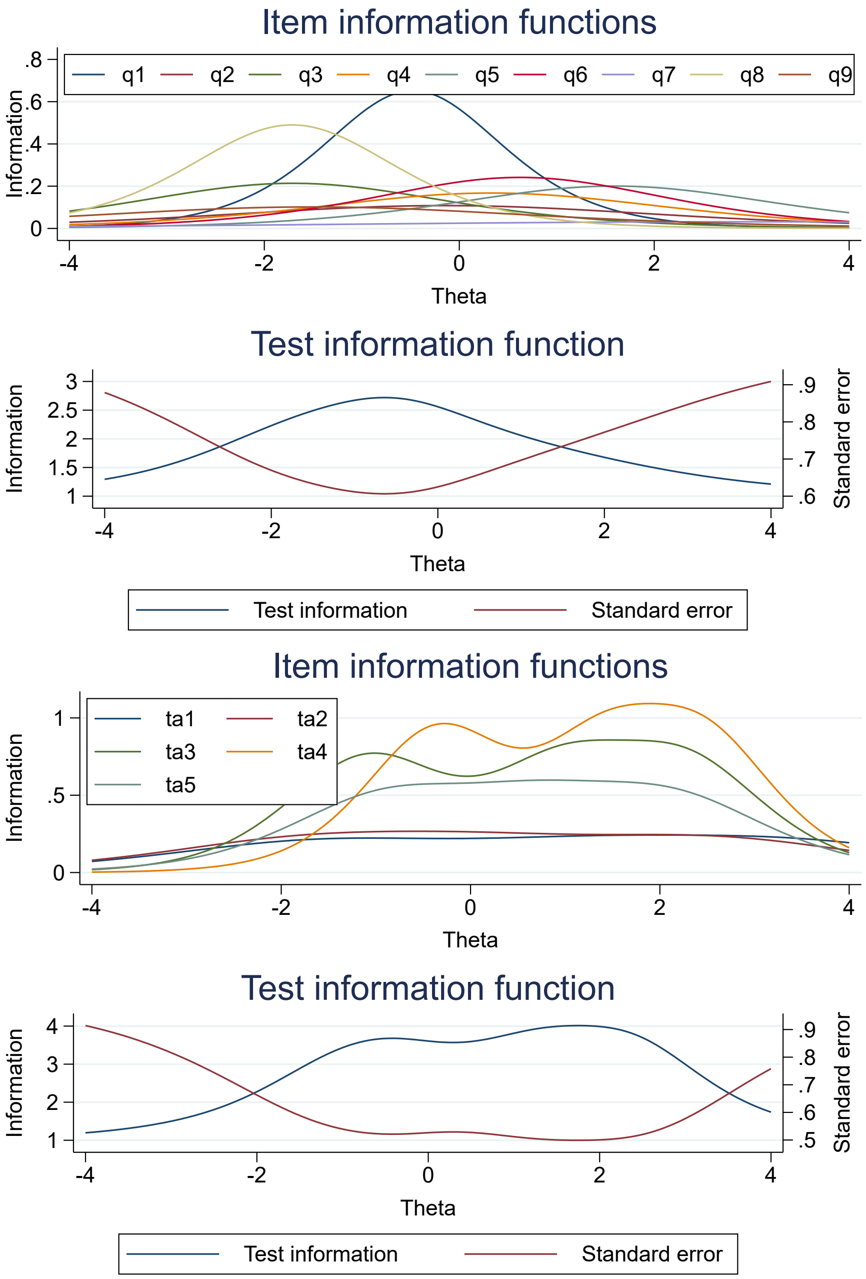
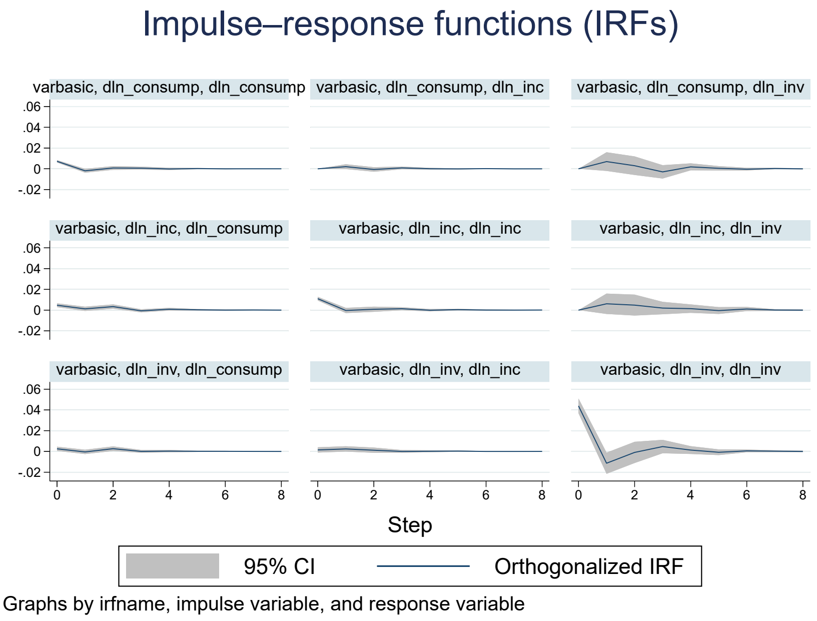


Figure 11.8. IRT graphs



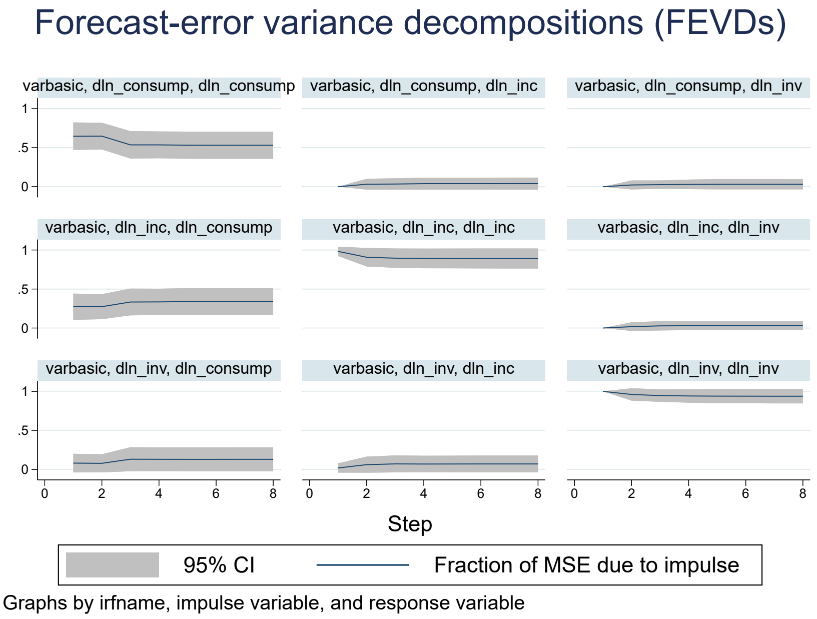


Figure 11.9. IRF and FEVD graphs after VAR

11.2 Common options for statistical graphs

This section illustrates how to use Stata graph options with specialized statistical graph commands. Many of the examples will assume that you have run the command

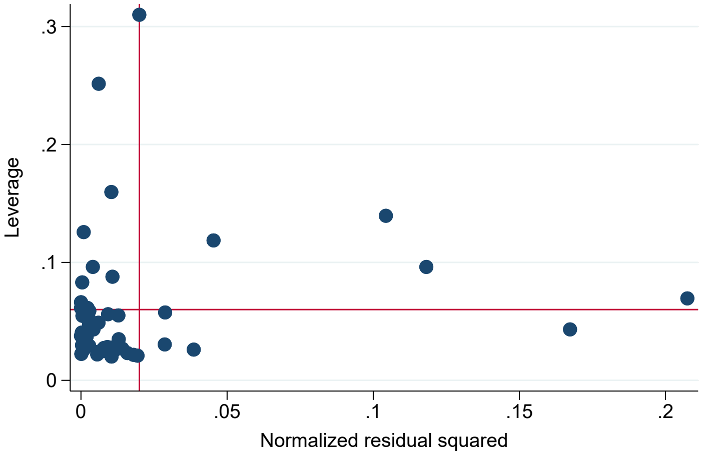
```
. regress propval100 popden pcturban
```

and will illustrate subsequent commands with options to customize those specialized statistics graphs.

```
lvr2plot
```

Consider this regression analysis, which predicts `propval100` from two variables, `popden` and `pcturban80`. The `lvr2plot` command produces a leverage-versus-residual squared plot. *Uses allstates.dta*

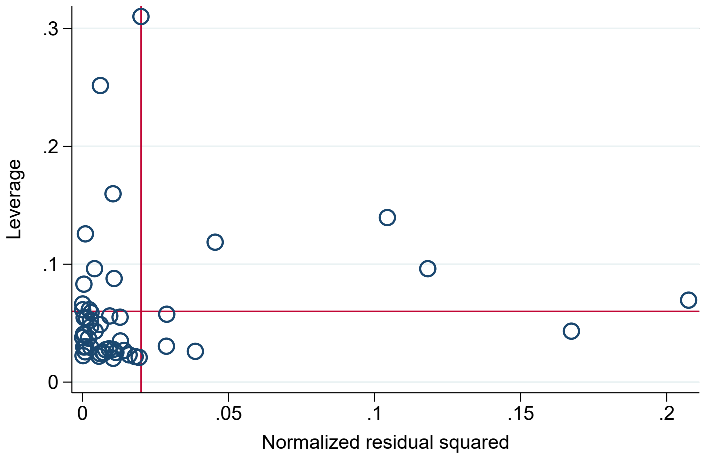
Before running the graph command, type
`reg propval100 popden pcturban80`



```
lvr2plot, msymbol(Oh) msize(vlarge)
```

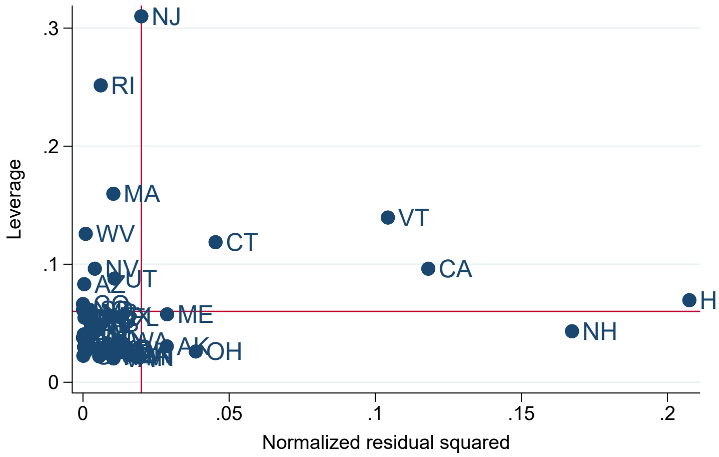
Here we add the `msymbol()` and `msize()` options to control the display of the markers in the graph. See Options : Markers (section 8.1) for more details. *Uses allstates.dta*

Before running the graph command, type
`reg propval100 popden pcturban80`



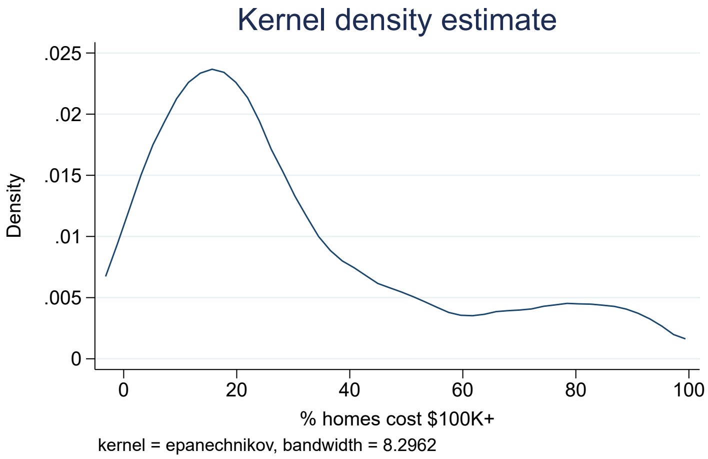
```
lvr2plot, mlabel(stateab)
```

The `mlabel()` option adds marker labels to the graph. We could also add more options to control the size, color, and position of the marker labels; see Options : Marker labels (section 8.2) for more details. *Uses allstates.dta Before running the graph command, type* `reg propval100 popden pctorban80`



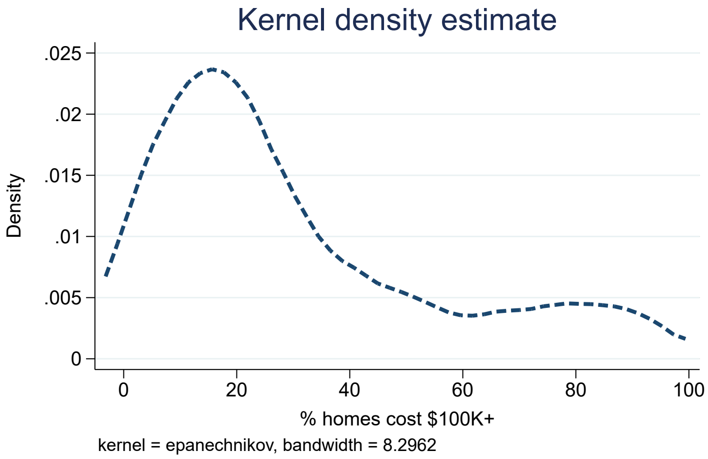
```
kdensity propval100
```

Consider this kernel density plot for the variable `propval100`. We could add options to control the display of the line. See the following example. *Uses allstates.dta*



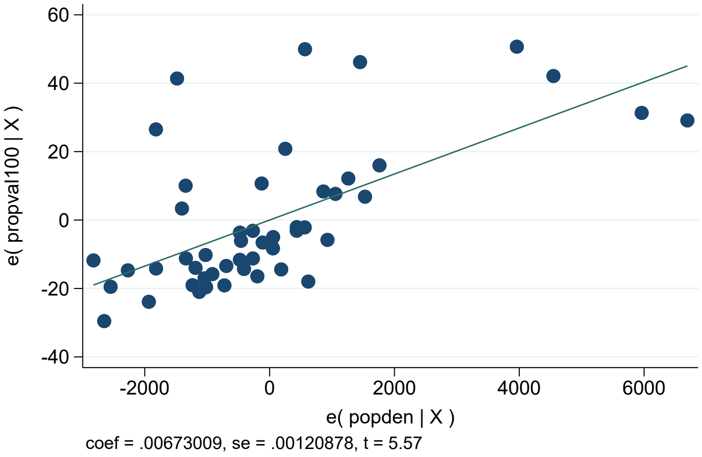
```
kdensity propval100, lwidth(thick) lpattern(dash)
```

The section **Options : Connecting** (section 8.3) shows several options we could add to control the display of the line. Here we add the `lwidth()` and `lpattern()` options to make the line thick and dashed. *Uses allstates.dta*

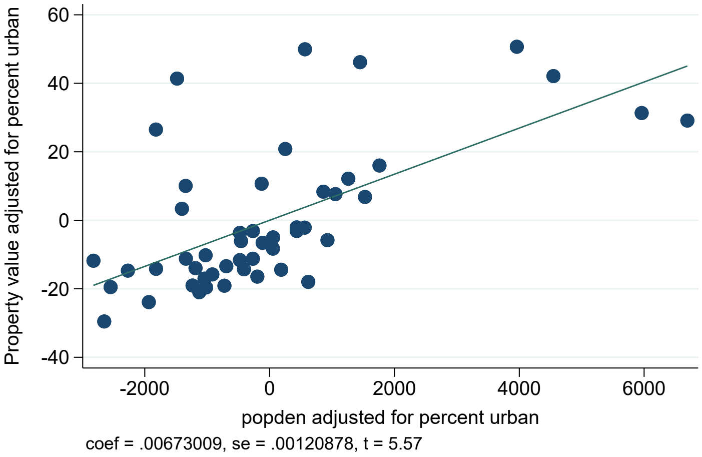


```
avplot popden
```

Consider this added-variable plot. We can modify the axes titles as illustrated in the following examples. *Uses allstates.dta*
Before running the graph command, type
`reg propval100 popden pcturban80`



```
avplot popden, xtitle("popden adjusted for percent urban")
ytitle("Property value adjusted for percent urban")
```



Here we use the `xtitle()` and `ytitle()` options to change the titles of the x and y axes. See Options : Axis titles (section 8.4) for more details. *Uses allstates.dta*

Before running the graph command, type

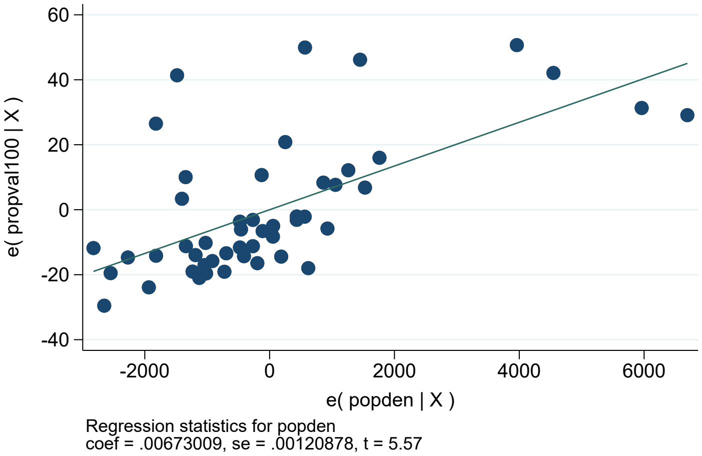
```
reg propval100 popden pcturban80
```

```
avplot popden, note("Regression statistics for popden",
prefix)
```

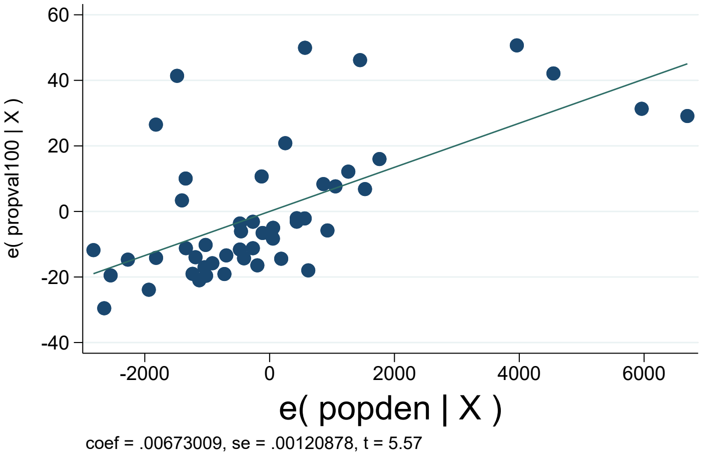
Here we use `prefix` within the `note()` option to add text before the existing note. We can do likewise for an existing title, subtitle, or caption. We could also use the `suffix` option to add information after an existing title. *Uses allstates.dta*

Before running the graph command, type

```
reg propval100 popden pcturban80
```



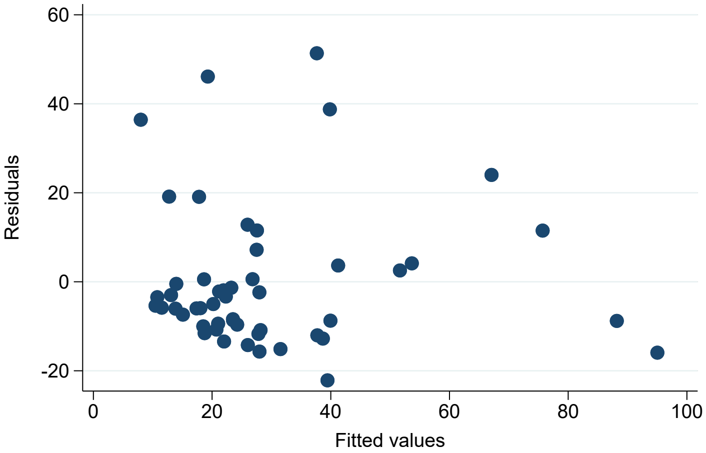
```
avplot popden, xtitle(, size(huge))
```



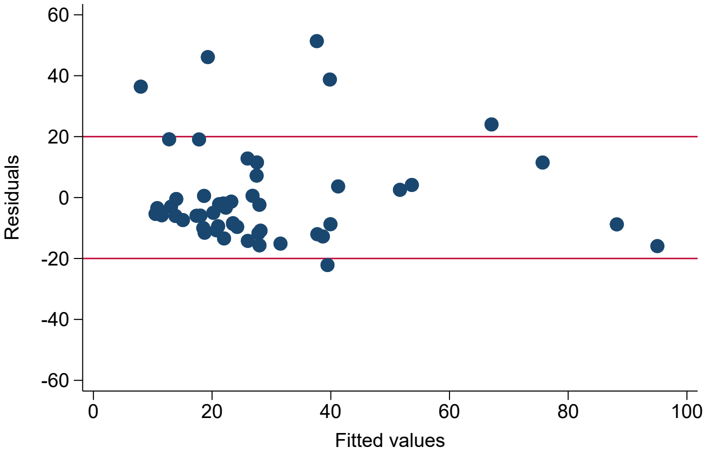
We can modify the look of the existing title without changing the text. Here we add the `size(huge)` option to make the existing title huge. See Options : Axis titles (section 8.4) and Options : Textboxes (section 8.11) for more details. *Uses allstates.dta*
Before running the graph command, type
`reg propval100 popden pcturban80`

```
rvfplot
```

Consider this residual-versus-fit plot. We often hope to see an even distribution of points around zero on the y axis. To help evaluate this distribution, we might want to label the y axis identically for the values above 0 and below 0. *Uses allstates.dta*
Before running the graph command, type
`reg propval100 popden pcturban80`



```
rvfplot, ylabel(-60(20)60, nogrid) yline(-20 20)
```

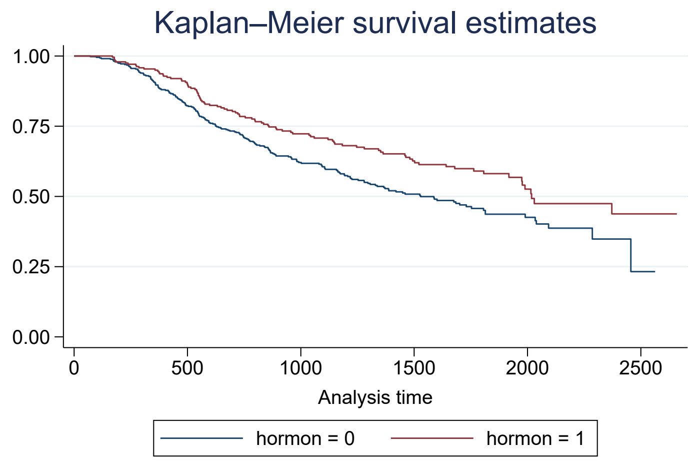


Here we add the `ylabel()` option to label the y axis from -60 to 60, incrementing by 20, and suppress the grid. Further, we use the `yline()` option to add a y line at 20 and -20. For more information about labeling and scaling axes, see Options : Axis labels (section 8.5) and Options : Axis scales (section 8.6). *Uses allstates.dta*

Before running the graph command, type
`reg propval100 popden pcturban80`

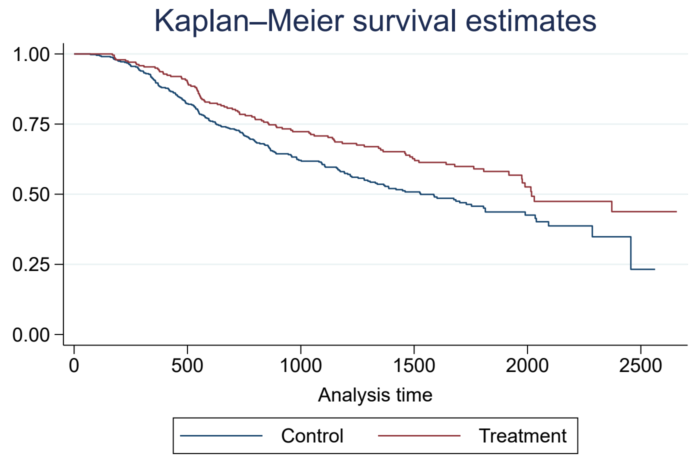
```
sts graph, by(hormon)
```

This graph shows survival-time estimates broken down by whether one is in the treatment group or the control group. The legend specifies the groups, but we might want to modify the labels as shown in the next example. *Uses hormone.dta*



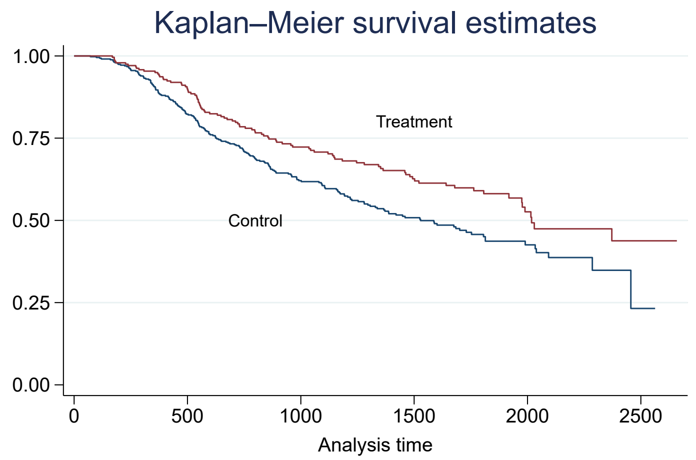
```
sts graph, by(hormon) legend(label(1 Control) label(2
Treatment))
```

We can use the `legend()` option to use different labels within the legend. See **Options : Legend** (section 8.9) for more details. *Uses hormone.dta*



```
sts graph, by(hormon) legend(off) text(.5 800 "Control")
text(.8 1500 "Treatment")
```

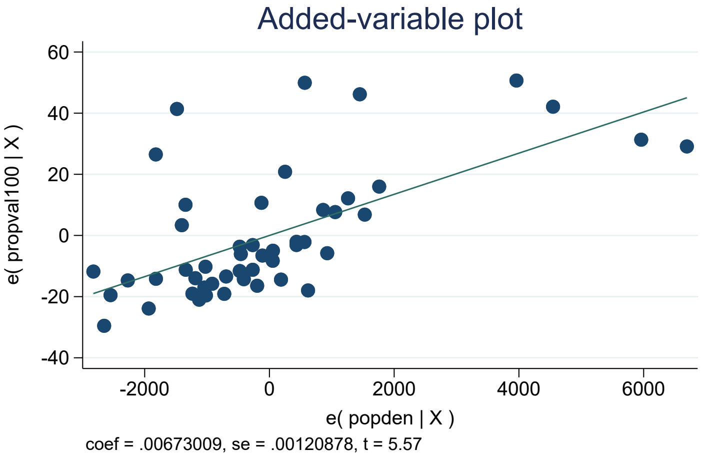
We use the `legend(off)` option to suppress the display of the legend and use the `text()` option to add text directly to the graph to label the two lines; see **Options : Adding text** (section 8.10) for more information. *Uses hormone.dta*



```
avplot popden, title("Added-variable plot")
```

We return to the regression analysis predicting `propval100` from `popden` and `pcturban80`. We add a title by using the `title()` option, but we could also add a `subtitle()`, `caption()`, or `note()`. *Uses allstates.dta*

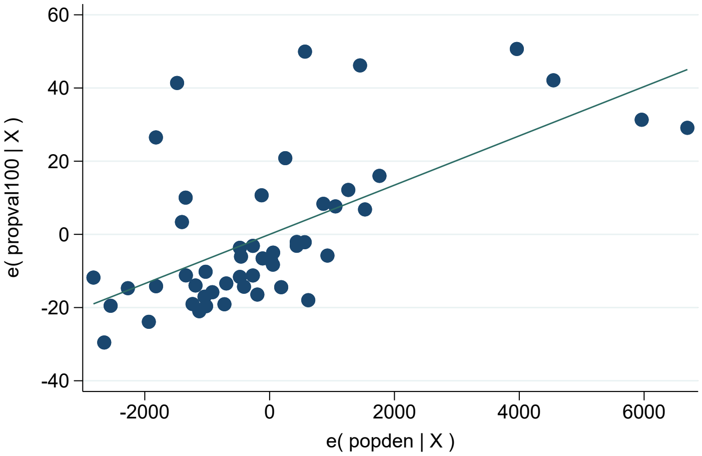
Before running the graph command, type `reg propval100 popden pcturban80`



```
avplot popden, note("")
```

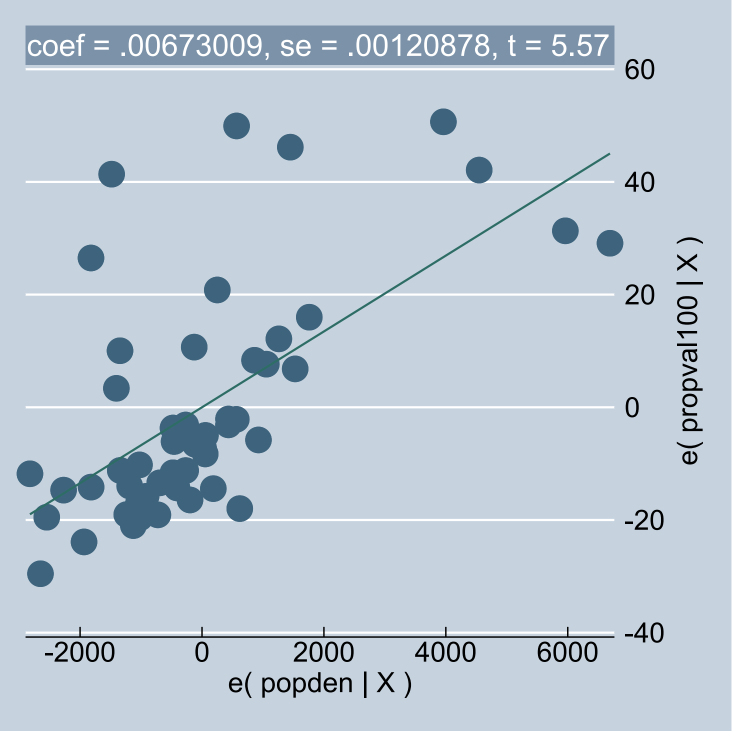
Here we add the `note("")` option, which suppresses the display of the note at the bottom showing the coefficients for the regression model. *Uses allstates.dta*

Before running the graph command, type
`reg propval100 popden pcturban80`



```
avplot popden, scheme(economist)
```

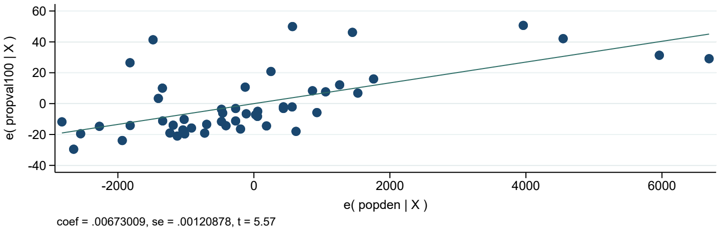
We can change the look of the graph by selecting a different scheme. Here we use `scheme(economist)` to display the graph using the `economist` scheme. See Standard options : Schemes (section 9.2) for more details. *Uses allstates.dta*
Before running the graph command, type `reg propval100 popden pcturban80`



```
avplot popden, xsize(3) ysize(1) scale(1.3)
```

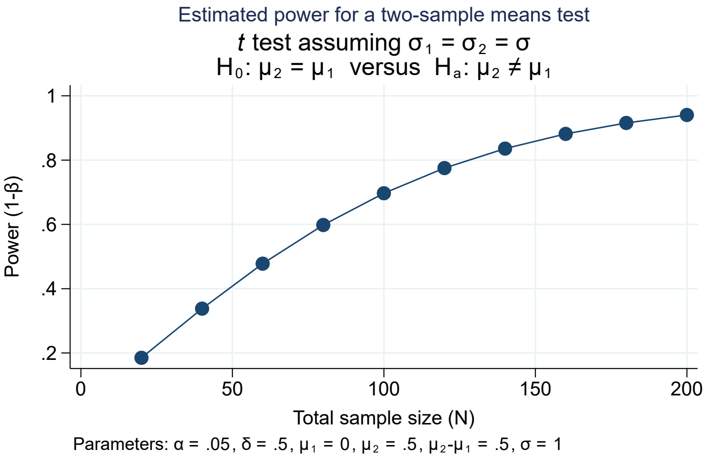
The section Standard options : Sizing graphs (section 9.3) describes options we can use to control the size of the graph and the scale of the contents of the graph. Here we show the `xsize()`, `ysize()`, and `scale()`

options. *Uses allstates.dta* *Before running the graph command, type* `reg propval100 popden pcturban80`



The next set of examples illustrate the creation and customization of graphs using the `power` command. Specifically, the examples will use the `power twomeans` command; however, the graph customizations I demonstrate would apply to most of, if not all, the `power` family of commands.

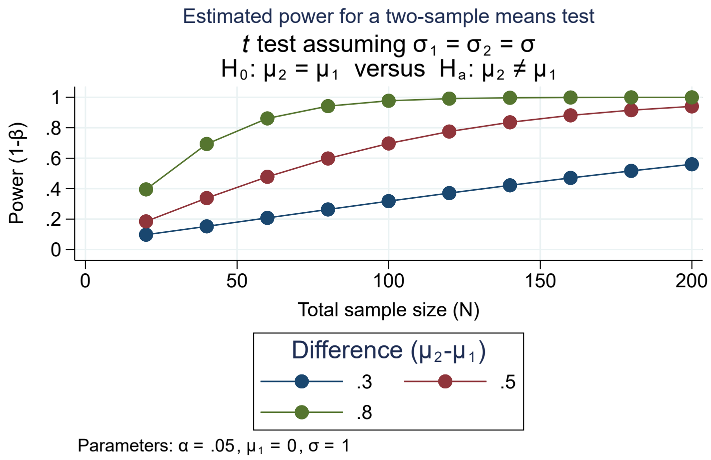
```
power twomeans 0, sd(1) n(20(20)200) diff(0.5) graph
```



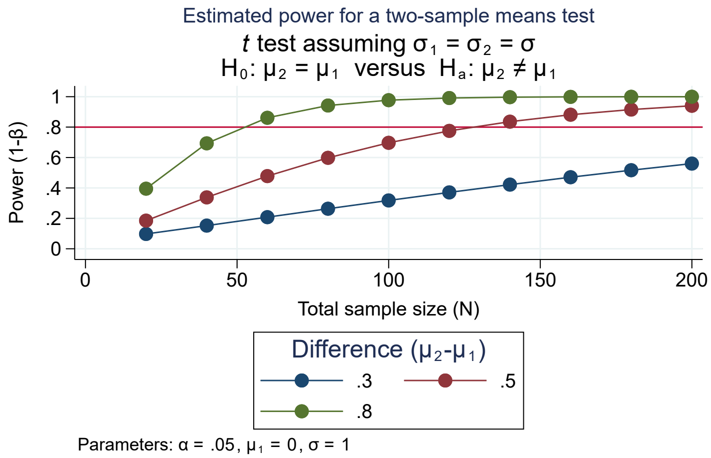
Consider this graph showing the power of a two-group t test. Using the `n(20(20)200)` option, we compute power for sample sizes (N) ranging from 20 to 200. The `diff(0.5)` option specifies the difference in means for the power analysis, in this case a difference of `0.5`. Note that we specified the standard deviation (for each group) as 1 via the `sd(1)` option. This graph shows, for a standardized difference of half a standard deviation, the power on the y axis as a function of the sample sizes shown on the x axis. *Uses allstates.dta*

```
power twomeans 0, sd(1) n(20(20)200) diff(0.3 0.5 0.8) graph
```

We can specify multiple values within the `diff()` option. By using the `diff(0.3 0.5 0.8)` option, we obtain power analyses assuming a difference of 0.3 units, 0.5 units, and again a difference of 0.8 units. The graph includes a separate line for each difference, labeled using the legend. For each line, we see the power on the y axis as a function of the sample sizes on the x axis. *Uses allstates.dta*



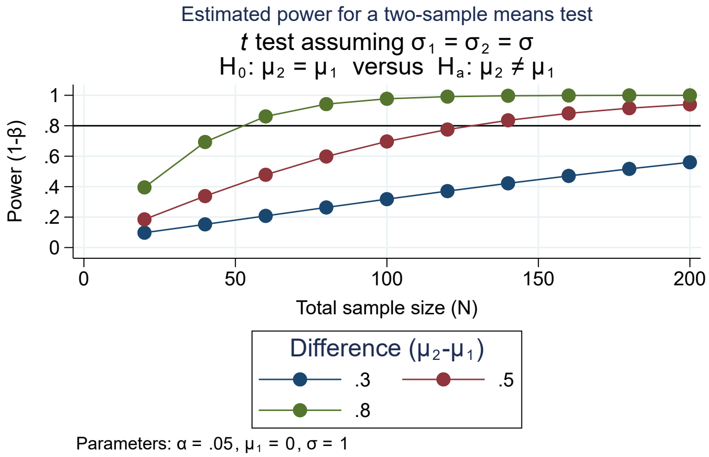
```
power twomeans 0, sd(1) n(20(20)200) diff(0.3 0.5 0.8)
graph(yline(0.8))
```



We would like to show a y line at 0.8 to call attention to the combinations of values that yield power of 0.8 or greater. In a normal `twoway` graph, we would achieve this by specifying `yline(0.8)`. In the context of the `power` command, we specify `graph(yline(0.8))`. Note how my options customizing the graph sit inside the `graph()` option. We can (and will) insert lots of options inside the `graph()` option. *Uses allstates.dta*

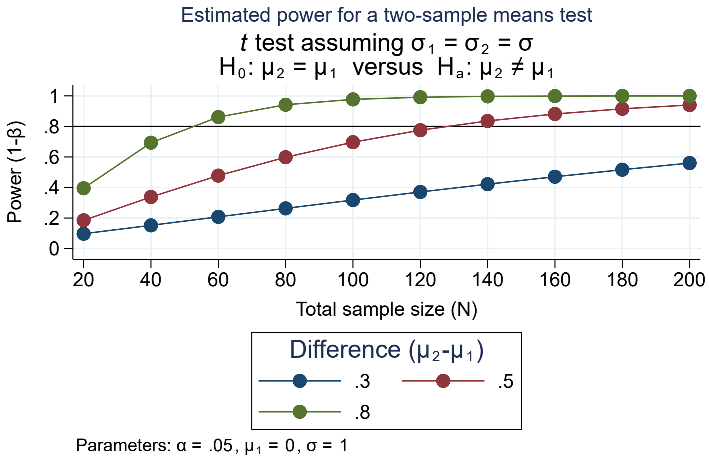
```
power twomeans 0, sd(1) n(20(20)200) diff(0.3 0.5 0.8)
graph(yline(0.8, lcolor(black)))
```

We want to use `black` for the color of the line drawn via the `yline()` option. We can make that change by adding `lcolor(black)` within the `graph(yline())` option. *Uses allstates.dta*



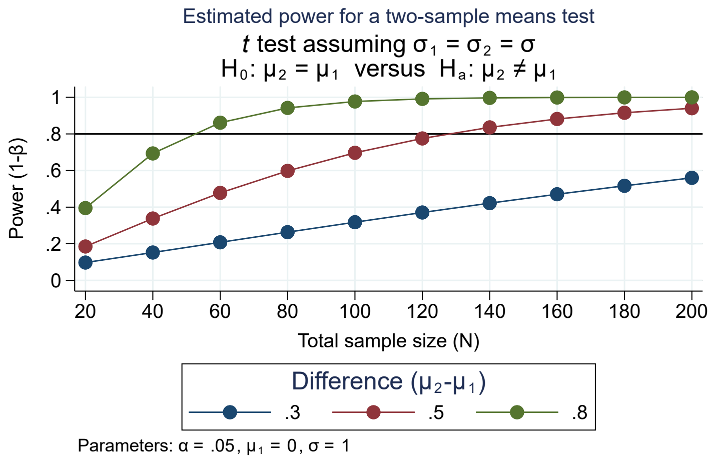
```
power twomeans 0, sd(1) n(20(20)200) diff(0.3 0.5 0.8)
graph(yline(0.8, lcolor(black)) xlabel(20(20)200))
```

We would like the labels for the x axis to mirror the sample sizes we specified. We can make that change easily by adding the `xlabel(20(20)200)` suboption within the `graph()` option. Now the x-axis labels mirror the sample sizes we specified. *Uses allstates.dta*



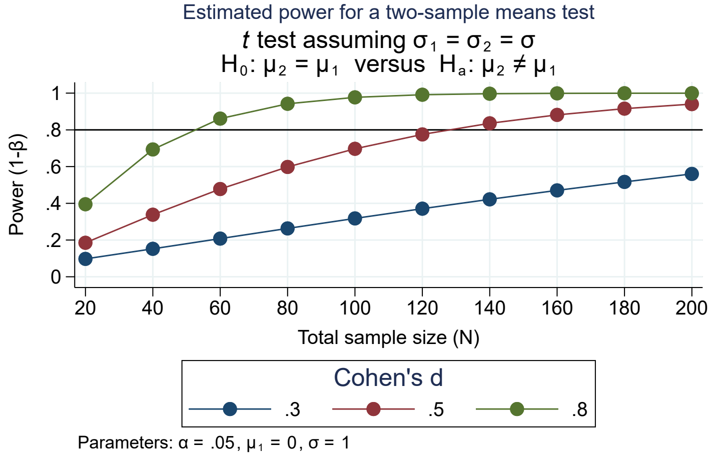
```
power twomeans 0, sd(1) n(20(20)200) diff(0.3 0.5 0.8)
graph(yline(0.8, lcolor(black)) xlabel(20(20)200)
legend(rows(1)))
```

We would prefer to display the legend using one row. We add the `legend(rows(1))` suboption within the `graph()` option, and the legend is now shown in one row. *Uses allstates.dta*



```
power twomeans 0, sd(1) n(20(20)200) diff(0.3 0.5 0.8)
graph(yline(0.8, lcolor(black)) xlabel(20(20)200)
legend(rows(1) title("Cohen’s d")))
```

We want to call attention to the fact that the difference in means is a standardized effect size, also called “Cohen’s *d*”. When we add `title("Cohen’s d")` within the `graph(legend())` option, the title of the legend says `Cohen’s d`. *Uses allstates.dta*

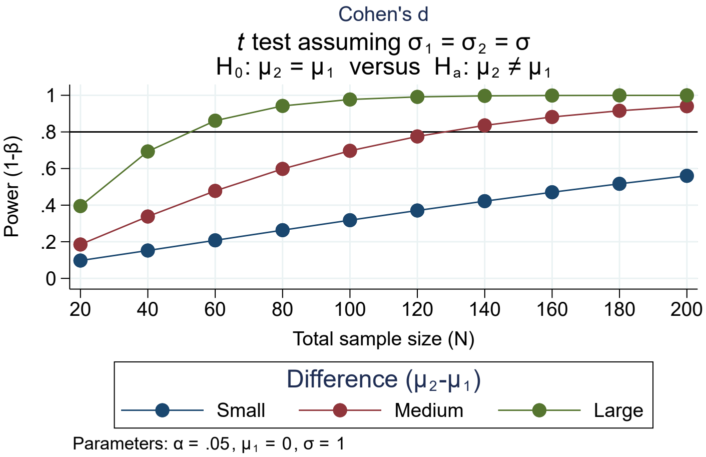


```
power twomeans 0, sd(1) n(20(20)200) diff(0.3 0.5 0.8)
graph(yline(0.8, lcolor(black)) xlabel(20(20)200)
legend(rows(1)) title("Cohen’s d") plotdim( , labels("Small"
"Medium" "Large")))
```

Cohen characterized an effect size of 0.3 as *Small*, 0.5 as *Medium*, and 0.8 as *Large*. Note that these correspond to the differences in means we specified. This example uses the `plotdim( , labels("Small" "Medium" "Large")))` suboption to assign the label `Small` to the first plotted value, `Medium` to the second plotted value, and `Large` to the third plotted value.

This alters the labels shown in the legend, accordingly.

Uses allstates.dta



11.3 The marginsplot command

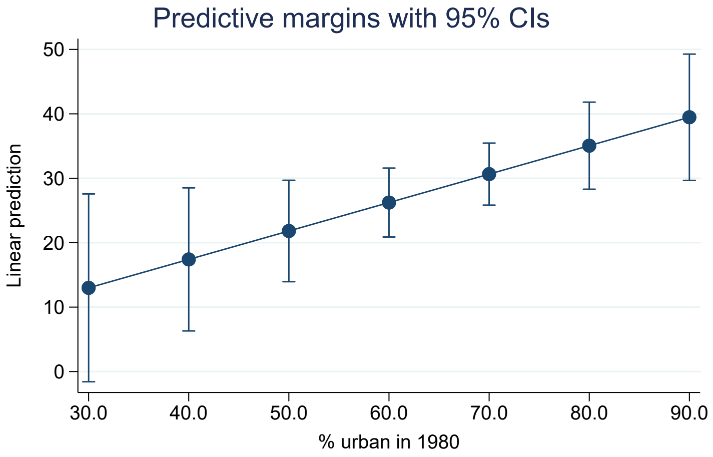
This section illustrates how to use the `marginsplot` command, including options that are common to all graph commands as well as options that are specific to `marginsplot`. The following examples will assume that you have run the following `regress` and `margins` commands:

```
. use allstates
. regress propval100 popden pcturban80
. margins, at(pcturban80=(30(10)90))
```

The `regress` command models property values as a function of population density and the percentage of the area considered to be urban. The `margins` command computes the predictive margins as a function of `pcturban80`.

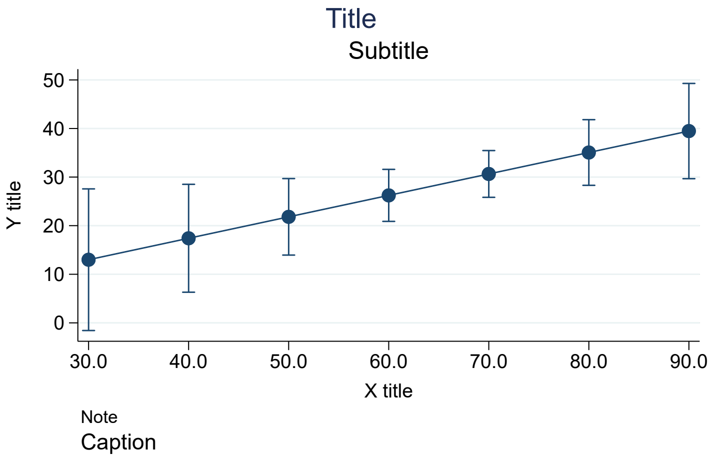
```
marginsplot
```

This graph shows the predictive margins as a function of `pcturban80`. The following examples illustrate how you can use options to customize this graph. *Uses allstates.dta*



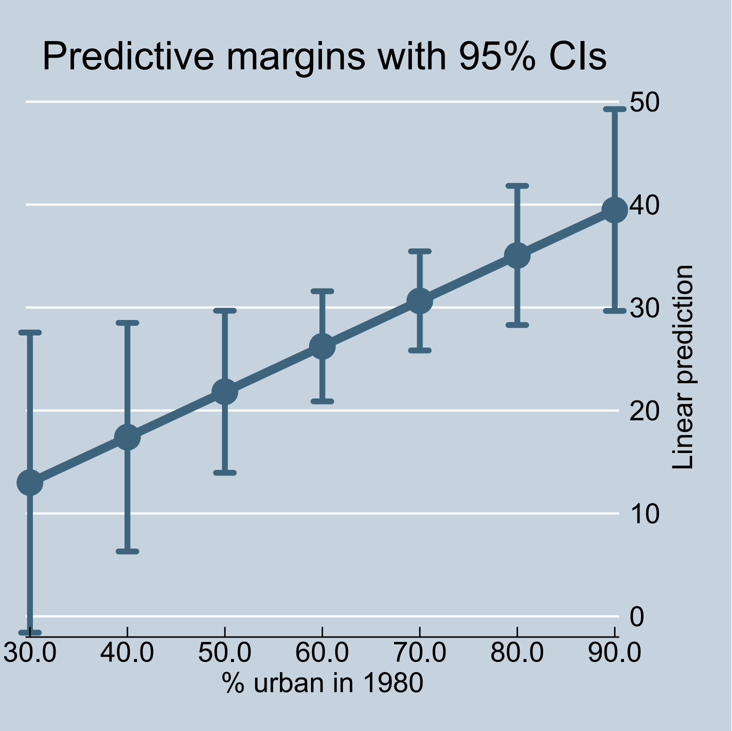
```
marginsplot, title(Title) subtitle(Subtitle) xtitle(X title)
ytitle(Y title) note(Note) caption(Caption)
```

You can add titles to the graph by using the `title()`, `subtitle()`, `xtitle()`, and `ytitle()` options. The `note()` and `caption()` options can also be used to annotate the graph. You can see Standard options : Titles (section 9.1) for more details about adding titles, notes, and captions. *Uses allstates.dta*



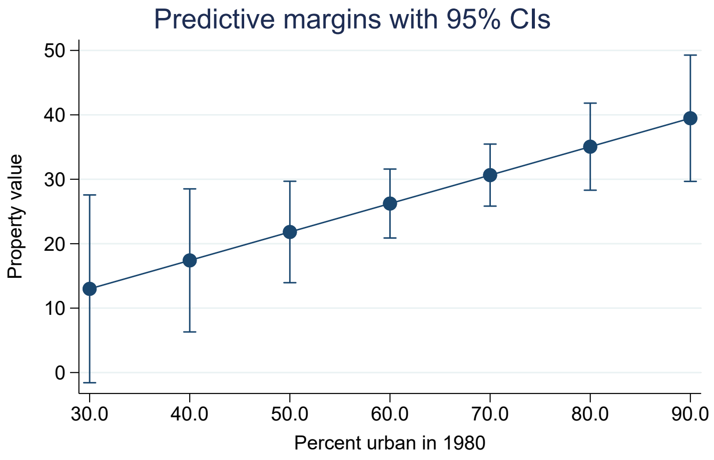
```
marginsplot, scheme(economist)
```

The `scheme()` option can be used to change the overall look of the graph. In this example, the `economist` scheme is used. See **Standard options : Schemes** (section 9.2) for more information about the selection of schemes. *Uses allstates.dta*



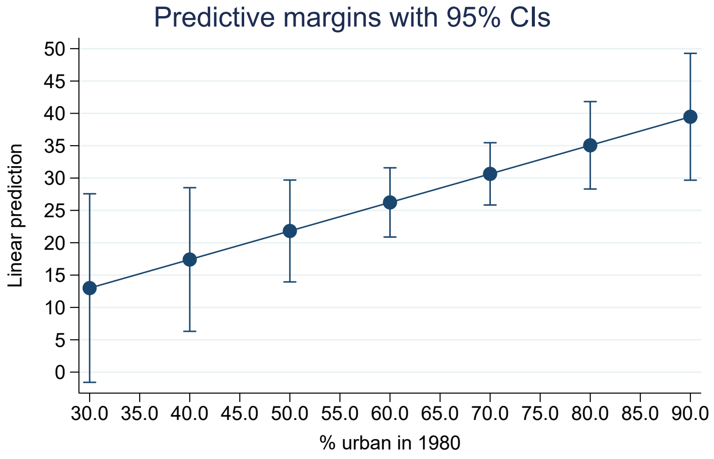
```
marginsplot, xtitle(Percent urban in 1980) ytitle(Property
value)
```

The `xtitle()` and `ytitle()` options are used to change the titles of the x and y axes. You can see **Options : Axis titles** (section 8.4) for more details about adding titles to axes. *Uses allstates.dta*



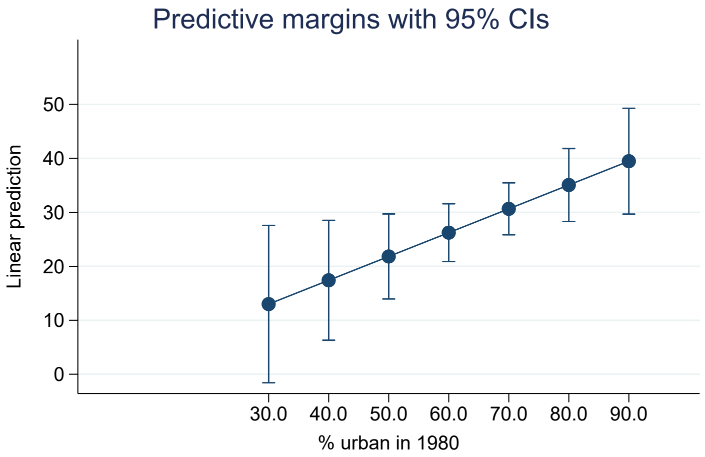
```
marginsplot, xlabel(30(5)90) ylabel(0(5)50)
```

The labeling of the x and y axes can be controlled with the `xlabel()` and `ylabel()` options. You can find more information about labeling axes in Options : Axis labels (section 8.5). *Uses allstates.dta*



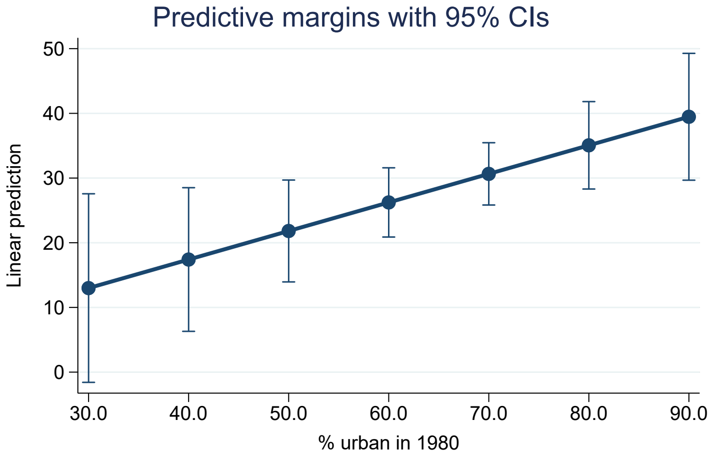
```
marginsplot, xscale(range(0 100)) yscale(range(0 60))
```

The `xscale()` and `yscale()` options can be used to expand the scale of the x and y axes. See Options : Axis scales (section 8.6) for more information about options that control the scale of the x and y axes. *Uses allstates.dta*



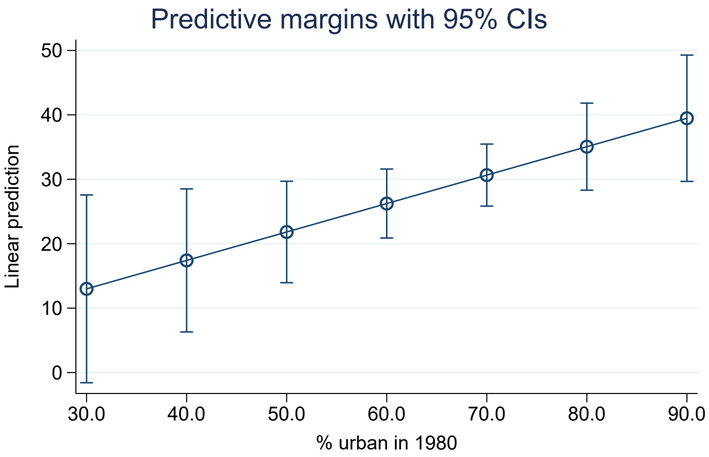
```
marginsplot, plotopts(clwidth(thick))
```

The `plotopts()` option allows you to include options that control the look of the line and markers. This example uses the `clwidth()` suboption to make the fitted line thick. You can see Styles : Linewidth (section 10.7) for more details about controlling the thickness of lines. *Uses allstates.dta*



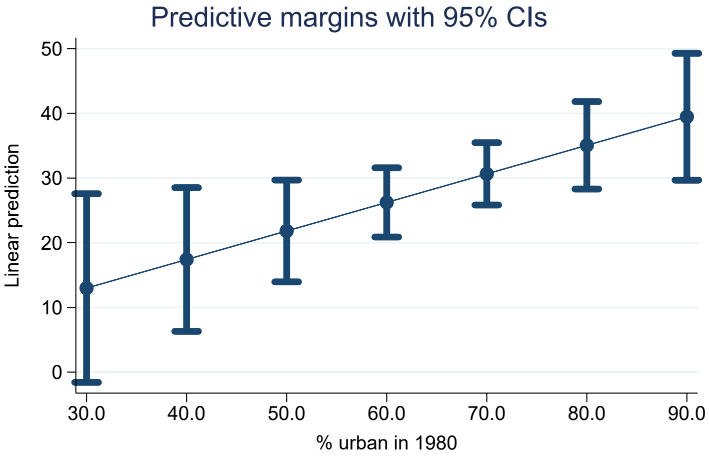
```
marginsplot, plotopts(msymbol(Oh) msize(large))
```

The `plotopts()` option is used in this example with the `msymbol()` and `msize()` suboptions to draw the markers as large hollow circles. For more information about selecting marker symbols, see Styles : Symbols (section 10.11). *Uses allstates.dta*



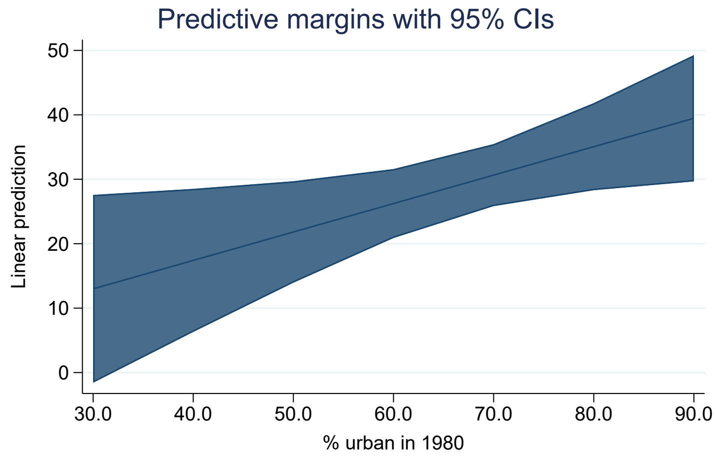
```
marginsplot, ciopts(lwidth(vthick) msize(huge))
```

The `ciopts()` option allows you to include options that control the look of the confidence interval. The `lwidth()` suboption makes the lines for the confidence intervals very thick, and the `msize()` suboption makes the cap of each confidence interval huge. *Uses allstates.dta*

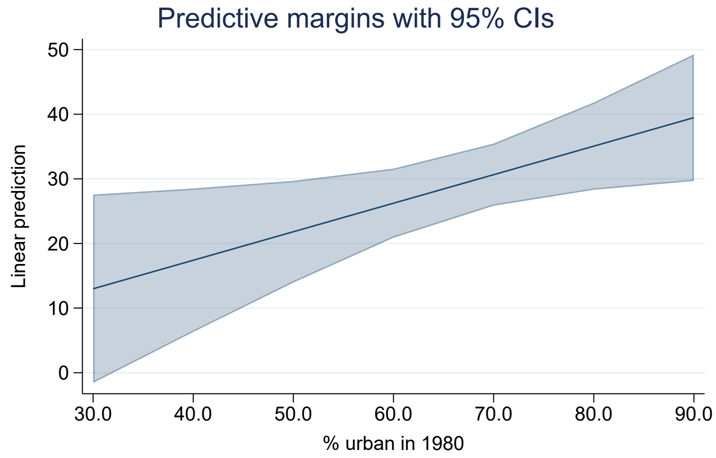


```
marginsplot, recast(line) recastci(rarea)
```

The `recast(line)` option specifies that the fitted line be drawn like a `twoway line` graph. The `recastci(rarea)` option specifies that the confidence interval be drawn like a `twoway rarea` graph. *Uses allstates.dta*



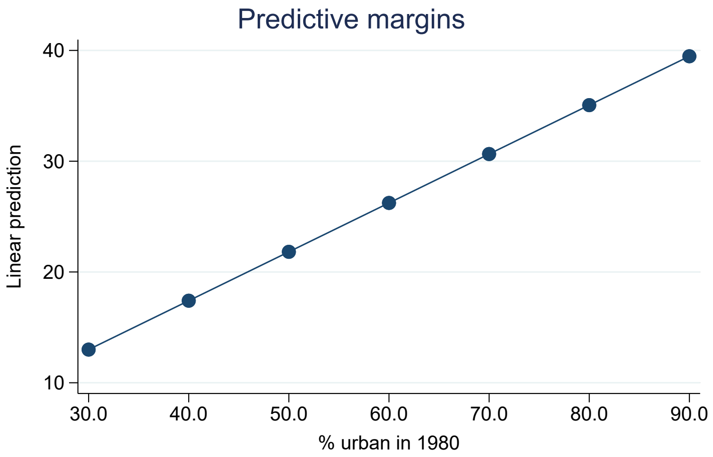
```
marginsplot, recast(line) recastci(rarea)
ciopt(color(navy%30))
```



Now that the graph is recast as an `rarea` plot, we can use the `ciopt(color(navy%30))` option to specify the color of the confidence region (and the line enclosing the confidence region). In this instance, the confidence region is shown using the color `navy` with 30% opacity. See Styles : Colors (section 10.2) for more details about colors (including intensity and opacity). *Uses allstates.dta*

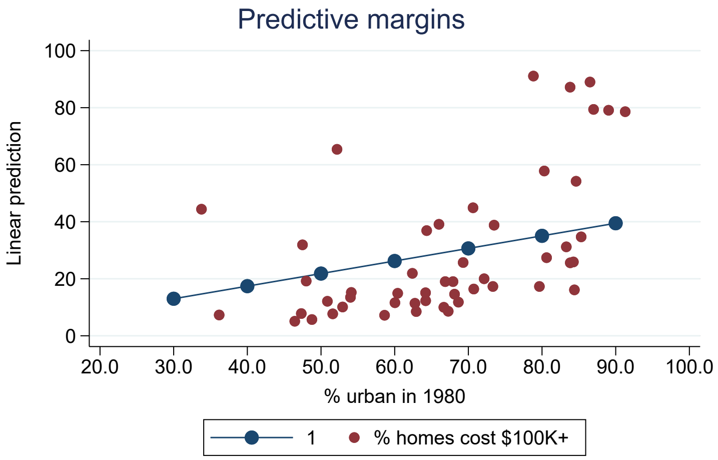
```
marginsplot, noci
```

The `noci` option suppresses the display of the confidence interval. *Uses allstates.dta*



```
marginsplot, noci addplot(scatter propval100 pcturban80,
msymbol(o))
```

The `addplot()` option can be used to overlay a new graph onto the graph created by the `margins` command. In this case, it overlays a scatterplot of `propval100` and `pcturban80`. *Uses allstates.dta*

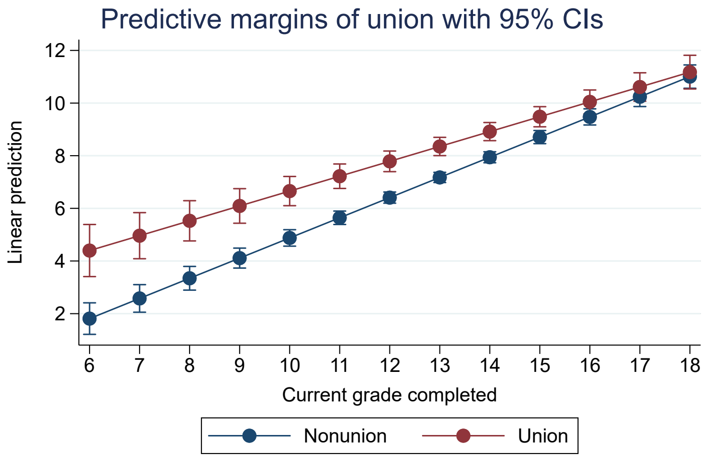


Let's now consider another example. We will use linear regression to model hourly `wage` as a function of `age`, education level (that is, `grade`), and `union` membership (1=yes, 0=no). The regression model also includes a `grade#union` interaction, which is significant. The commands below use `nlsw.dta` and run the `regress` command. To help us interpret the `grade` by `union` interaction, let's make a graph that shows the predictive margin of wages by `union` status and by `grade`. The first step is to use the `margins` command, shown below, to compute predictive margins by `union` status and `grade` (ranging from 6 to 18, in 1-unit increments).

```
. use nlsw, clear
. regress wage age c.grade##i.union
. margins union, at(grade=(6(1)18))
```

The `marginsplot` command can then be used to graph the predictive margins computed by the `margins` command.

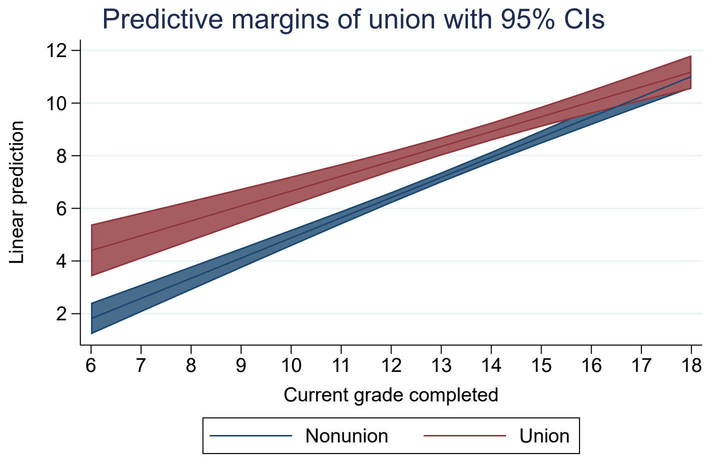
```
marginsplot
```



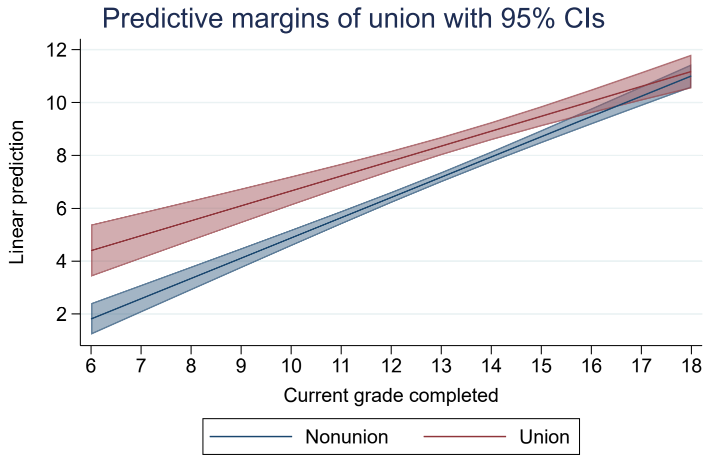
This example shows the graph created by the `marginsplot` command, illustrating the interaction between `grade` and `union`. One way to interpret the interaction is to note the wage gap between union and nonunion members grows smaller with increasing education. This might be easier to visualize if the confidence regions were shaded, as shown in the next example. *Uses nlsw.dta*

```
marginsplot, recast(line) recastci(rarea)
```

The `recast(line)` option is used to show the predictive margins as a `line` plot, and the `recastci(rarea)` option shows the confidence regions as an `rarea` plot. This provides a better visualization of the interaction of `grade` by `union`. Additionally, we can see the overlap of the two confidence regions. However, because of the opacity of the shading of the confidence regions, the overlap is not easy to see. *Uses nlsw.dta*



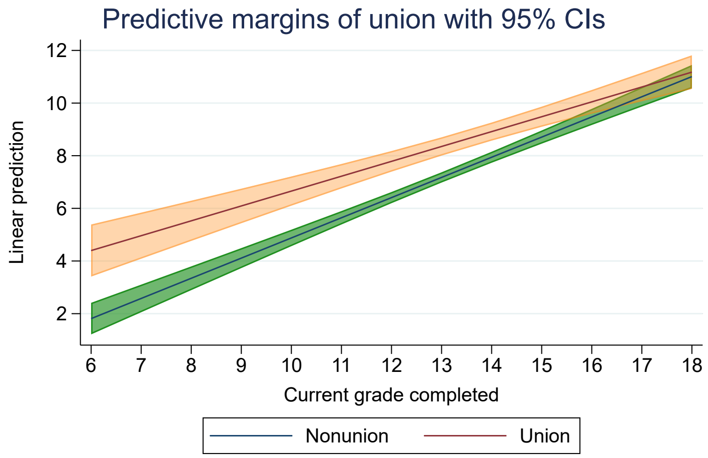
```
marginsplot, recast(line) recastci(rarea) ciopts(color(%50))
```



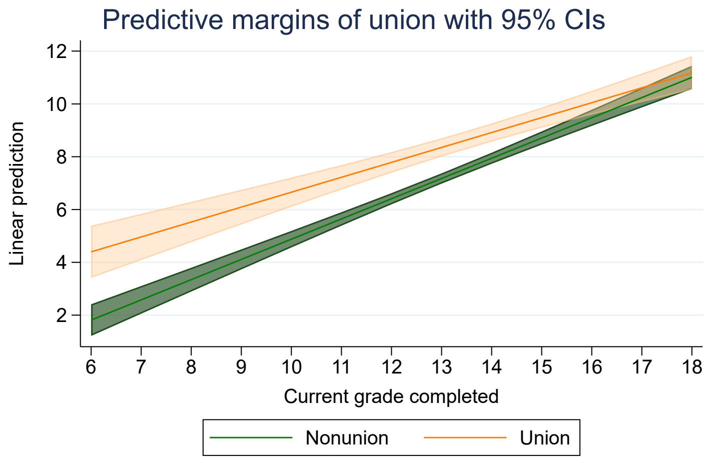
This example adds the `ciopts(color(%50))` option, which specifies the color used for displaying the confidence region. By specifying `color(%50)`, we use the default color but with 50% opacity. The overlap of the confidence regions is far easier to visualize by reducing the opacity of the color of the confidence regions. The next example shows how to specify custom colors for the confidence regions for nonunion and union members. *Uses nlsw.dta*

```
marginsplot, recast(line) recastci(rarea)
ci1opts(color(green%70)) ci2opts(color(orange%40))
```

In this example, we use the `ci1opts(color(green%70))` to specify the color of the confidence region for the first group. The confidence region for nonunion workers is shown in `green` with 70% opacity. The `ci2opts(color(orange%40))` to specifies the color of the confidence region for the second group. The confidence region for union workers is shown in `orange` with 40% opacity. The next example shows how to change the color of the fit line. *Uses nlsw.dta*



```
marginsplot, recast(line) recastci(rarea)
ci1opts(color(green*2.5%70)) plot1opts(lcolor(green))
ci2opts(color(orange*0.5%40)) plot2opts(lcolor(orange))
```



This example illustrates how you can modify the intensity (brightness) of the color of the confidence region for each group. For nonunion workers, the color of the confidence region is `green*2.5%70`, making the green a darker color than the prior example. For the union workers, the color of the confidence region is `orange*0.5%40`, making it brighter than the previous example. See **Styles : Colors** (section 10.2) for more details about colors (including intensity and opacity). *Uses nlsw.dta*

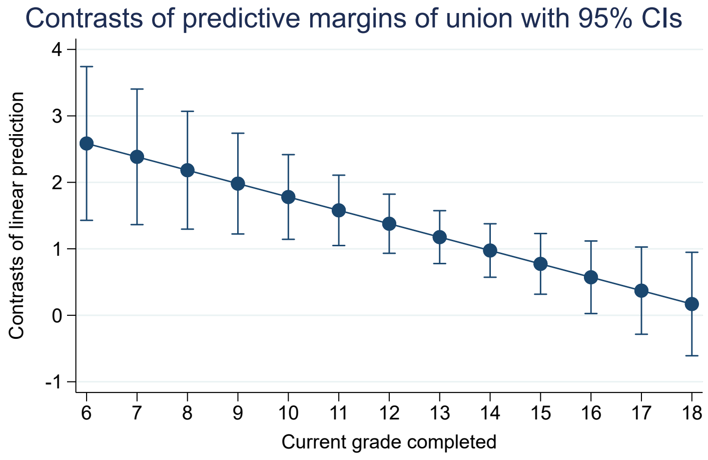
Another way to visualize the `grade`-by-`union` interaction is to use the `margins` command, shown below, to form a contrast between union and nonunion workers spanning the same levels of education (from 6 to 18 years).

```
. margins r.union, at(grade=(6(1)18))
```

The `marginsplot` command can then be used to graph the contrast and confidence interval computed by the `margins` command.

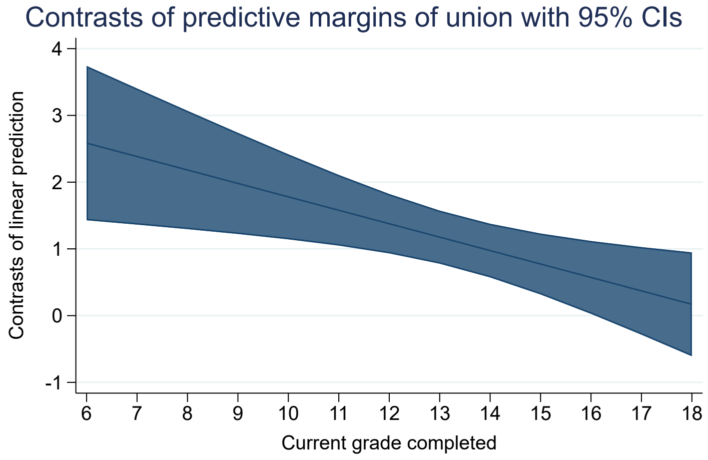
```
marginsplot
```

This example shows the graph created by the `marginsplot` command. The following examples illustrate how to customize this graph. *Uses nlsw.dta*



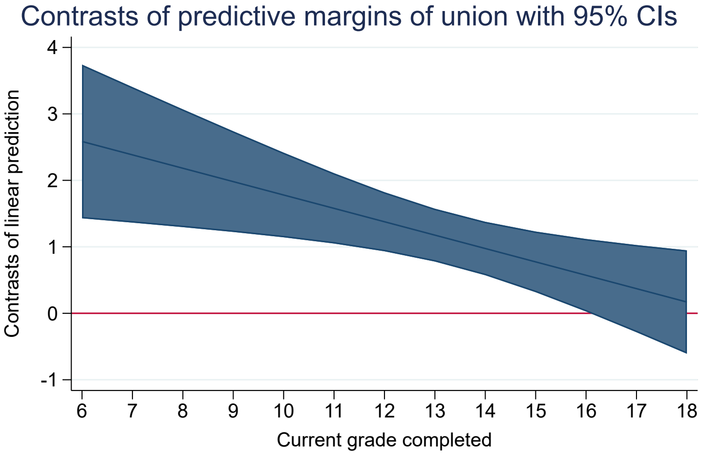
```
marginsplot, recast(line) recastci(rarea)
```

As we saw earlier, the `recast(line)` option is used to show the predicted values as a `line` plot, and the `recastci(rarea)` option is used to show the confidence region as an `rarea` plot. This allows us to visualize the difference and to note the areas where the confidence interval does not exclude 0. *Uses nlsw.dta*



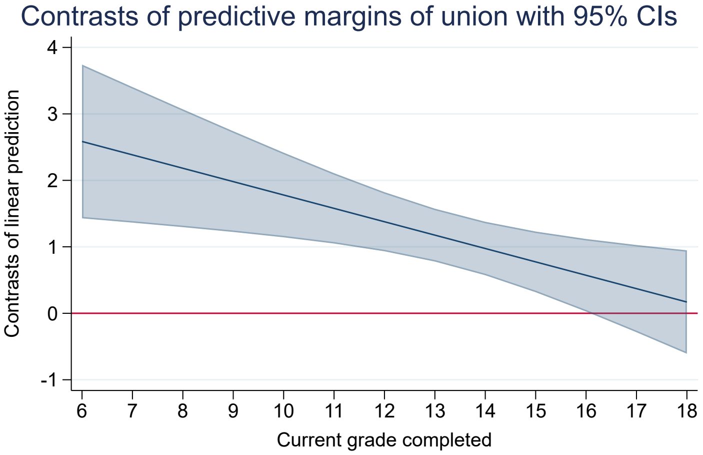
```
marginsplot, recast(line) recastci(rarea) yline(0)
```

To help emphasize the importance of the confidence intervals that include versus exclude 0, let’s add the `yline(0)` option. Where the confidence interval excludes 0, the difference between union and nonunion workers is significantly different. Unfortunately, this line is completely obscured when the confidence region overlaps the line. *Uses nlsw.dta*



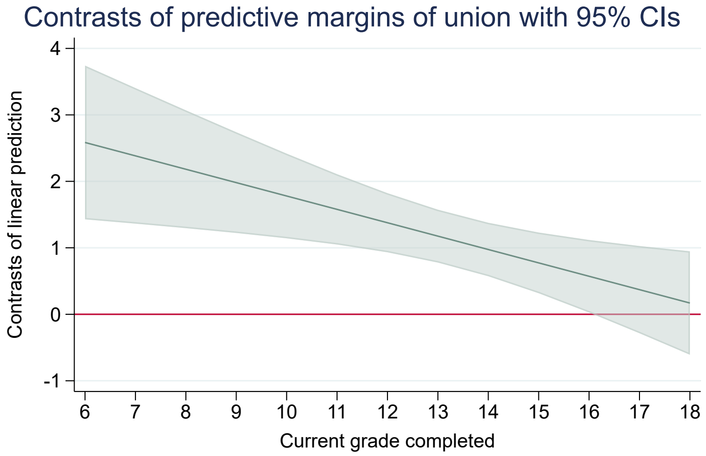
```
marginsplot, recast(line) recastci(rarea) yline(0)
ciopts(color(%30))
```

This example adds the `ciopts(color(%30))` option, which specifies the color used for displaying the confidence region. By specifying `color(%30)`, we use the default color but with 30% opacity. Now, the red line clearly shows through the confidence region. *Uses nlsw.dta*



```
marginsplot, recast(line) recastci(rarea) yline(0)
plotopts(lcolor(teal)) ciopts(color(teal*.5%50))
```

This example adds the `plotopts(lcolor(teal))` option to use the color `teal` for the predictive margin. Then, the `ciopts(color(teal*.5%50))` option is used for the color `teal*.5%50` for the confidence region; that is, `teal` with the intensity of `*.5` and 50% opacity. This results in a brighter and less opaque shade of teal. See Styles : Colors (section 10.2) for more details about colors (including intensity and opacity). *Uses nlsw.dta*



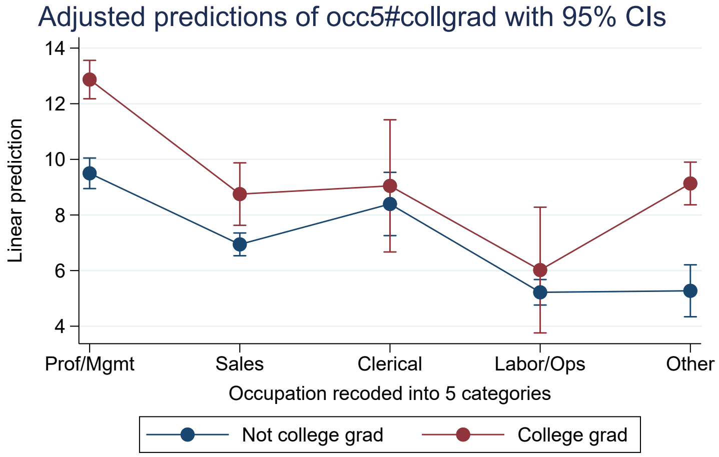
Let’s now consider another example, this one based on a 5×2 analysis of variance (ANOVA). The commands below use `nlsw.dta`, run the `anova` command, and then use the `margins` command to obtain the predictive margins by `occ5` and `collgrad`. The output of these commands is suppressed to save space.

```
. use nlsw, clear
. anova wage i.occ5##i.collgrad
. margins occ5#collgrad
```

The `marginsplot` command can then be used to graph the predictive margins computed by the `margins` command.

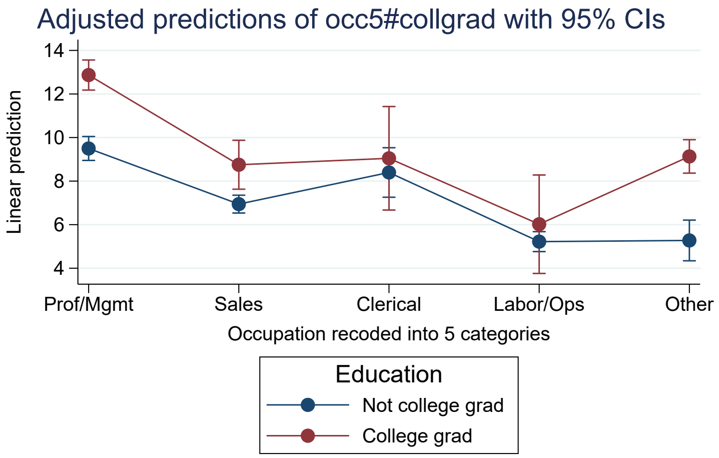
```
marginsplot
```

This example shows the graph created by the `marginsplot` command. The following examples illustrate how to customize this graph. *Uses nlsw.dta*



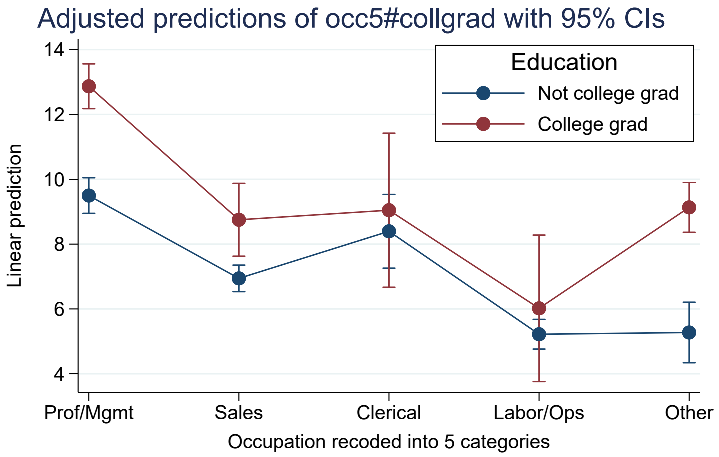
```
marginsplot, legend(subtitle("Education") rows(2))
```

This example includes the `legend()` option to customize the display of the legend, adding a subtitle and displaying the legend keys in two rows. You can see Options : Legend (section 8.9) for more details about customizing the legend. *Uses nlsw.dta*



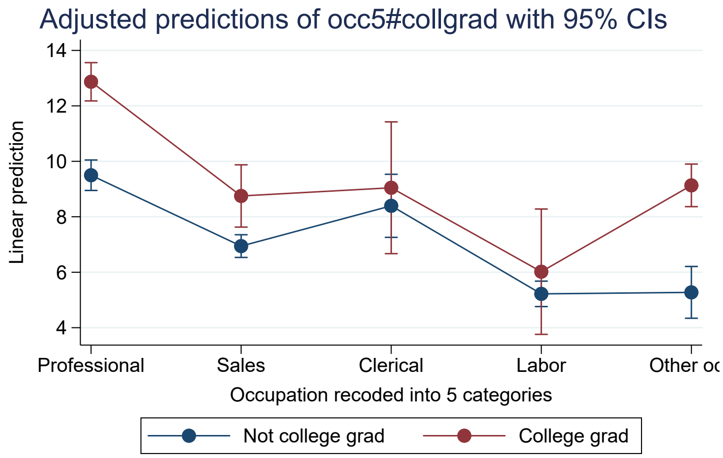
```
marginsplot, legend(subtitle("Education") rows(2) ring(0)
pos(1))
```

The `ring()` and `pos()` suboptions are added to the `legend()` option to display the legend within the graph in the 1 o'clock position. For more details about customizing the legend, see Options : Legend (section 8.9). *Uses nlsw.dta*



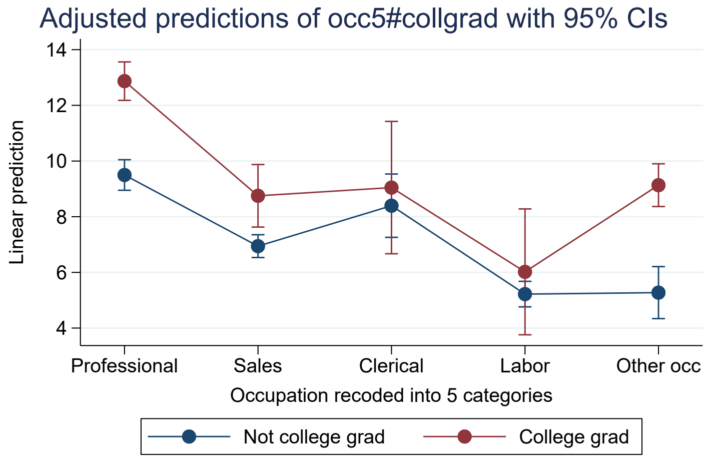
```
marginsplot, xlabel(1 "Professional" 2 "Sales" 3 "Clerical" 4
"Labor" 5 "Other occ")
```

The `xlabel()` option is used to control the labeling of the x axis, as shown in this example. See Options : Axis labels (section 8.5) for more information about axis labels. The next example illustrates how to address the issue of the label `Other occ` being cut off. *Uses nlsw.dta*



```
marginsplot, xlabel(1 "Professional" 2 "Sales" 3 "Clerical" 4
"Labor" 5 "Other occ") xscale(range(.75 5.25))
```

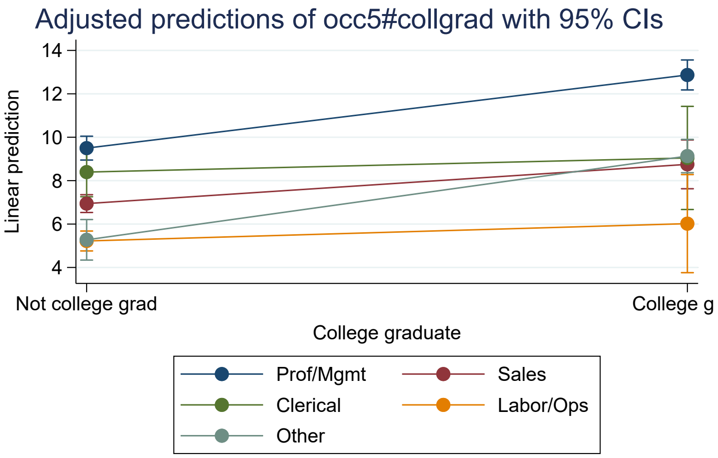
The `xscale(range())` option is used to expand the range of the x axis to make additional room for longer x-axis labels. You can see Options : Axis scales (section 8.6) for more information about controlling axis scales.
Uses nlsw.dta



In the previous examples, the variable `occ5` was placed on the x axis, and the variable `collgrad` was graphed using separate lines. Suppose that instead we want to place `collgrad` on the x axis.

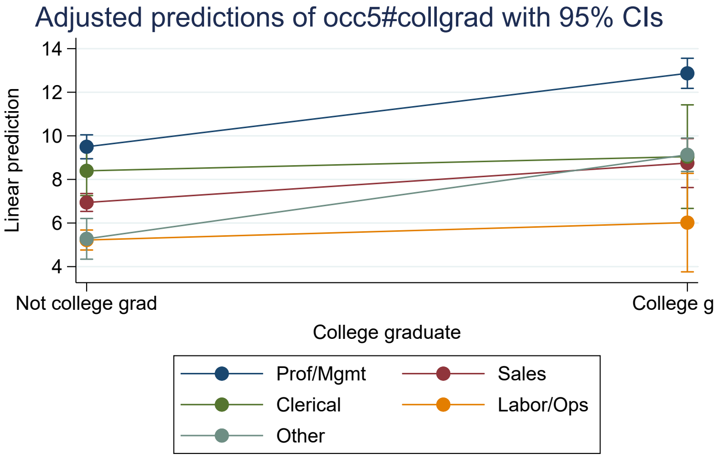
```
marginsplot, xdimension(collgrad)
```

The `xdimension()` option controls which variable is placed on the x axis. In this example, the variable `collgrad` is placed on the x axis. As a result, `collgrad` is graphed using separate lines. *Uses nlsw.dta*



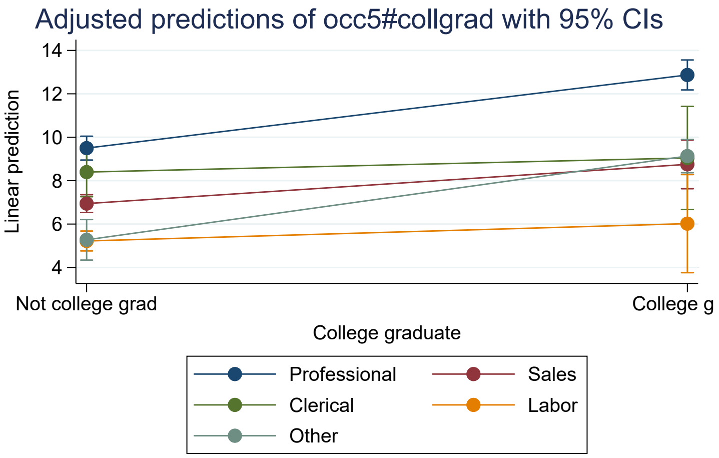
```
marginsplot, plotdimension(occ5)
```

The `plotdimension()` option controls which variable is graphed using the plot dimension, that is, graphed using separate lines. In this example, `occ5` is graphed using separate lines, and thus `collgrad` is placed on the x axis. *Uses nlsw.dta*

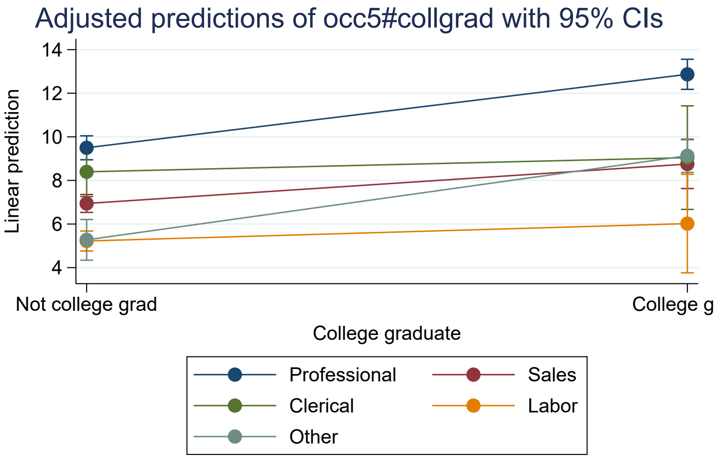


```
marginsplot, plotdim(occ5, labels("Professional" "Sales"
"Clerical" "Labor" "Other"))
```

The `labels()` suboption within the `plotdim()` option changes the labels used for the plot dimension. *Uses nlsw.dta*



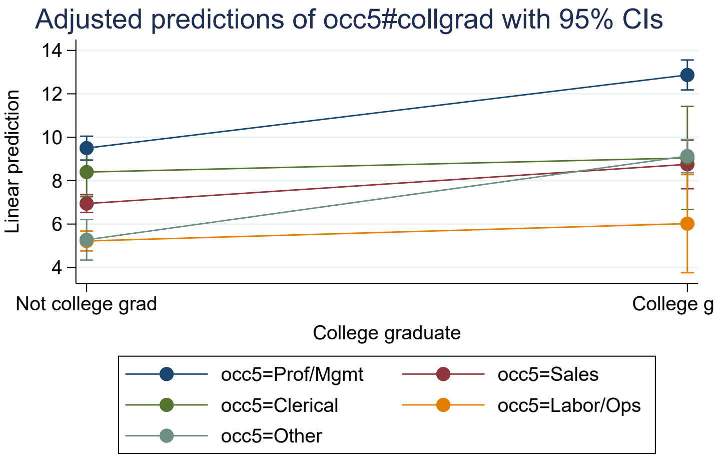
```
marginsplot, plotdim(occ5, elabels(1 "Professional" 4
"Labor"))
```



Using the `elabels()` suboption, you can selectively modify the labels of your choice. This example modifies the labels for the first and fourth occupations, leaving the labels for the other occupations unchanged. *Uses nlsw.dta*

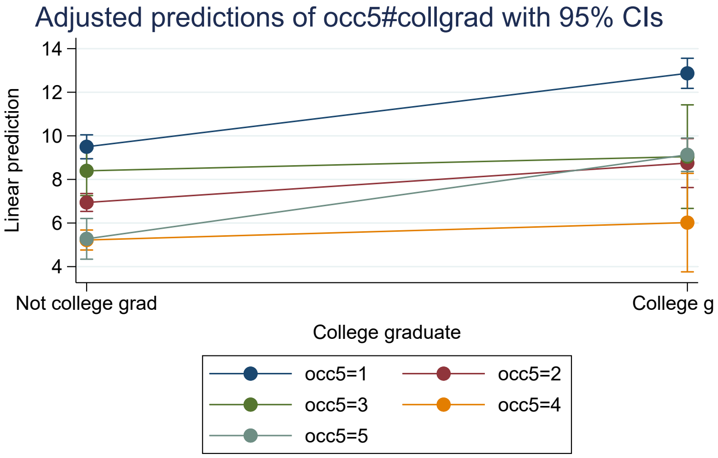
```
marginsplot, plotdim(occ5, nosimplelabels)
```

Adding the `nosimplelabels` suboption changes the plot label to the variable name, an equal sign, and the value label for the group. *Uses nlsw.dta*



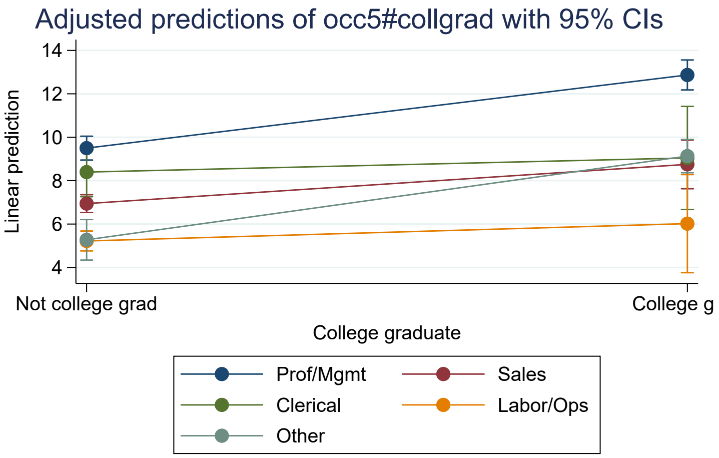
```
marginsplot, plotdim(occ5, nolabels)
```

Adding the `nolabels` suboption changes the plot label to the variable name, an equal sign, and the numeric value for the group. *Uses nlsw.dta*



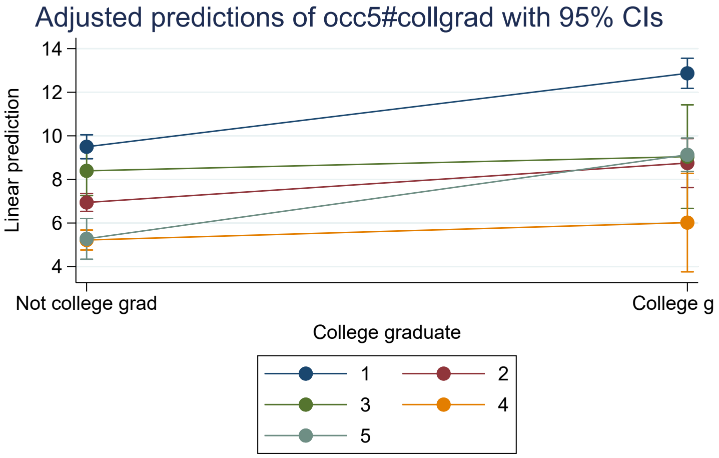
```
marginsplot, plotdim(occ5, allsimplelabels)
```

Using the `allsimplelabels` suboption yields a label that is composed solely of the value label for each group. *Uses nlsw.dta*



```
marginsplot, plotdim(occ5, allsimplelabels nolabels)
```

Using the `allsimplelabels` and `nolabels` suboptions displays a label that is composed solely of the numeric value for each group. *Uses nlsw.dta*



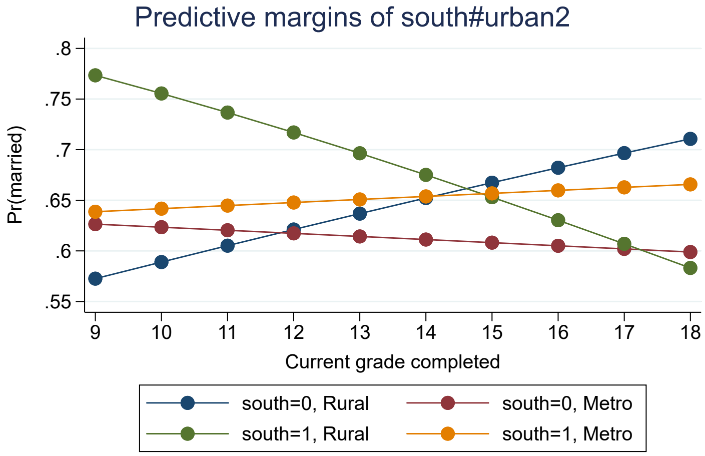
Let's nowconsider an example where the `marginsplot` command involves by-groups as an additional dimension. This example predicts marital status from the interaction of education, whether one lives in the South, and whether one lives in an urban area. The variables `race` and `age` are included as covariates. The command for running this logistic regression is shown below:

```
. use nlsw

. logit married c.grade##i.south##i.urban2 i.race age
```

The `c.grade#south#urban2` interaction is significant. (The output is omitted to save space.) The `margins` command (below) is used to compute the predictive margin of the probability of being married as a function of education, whether one lives in the South, and whether one lives in an urban area, after adjusting for race and age.

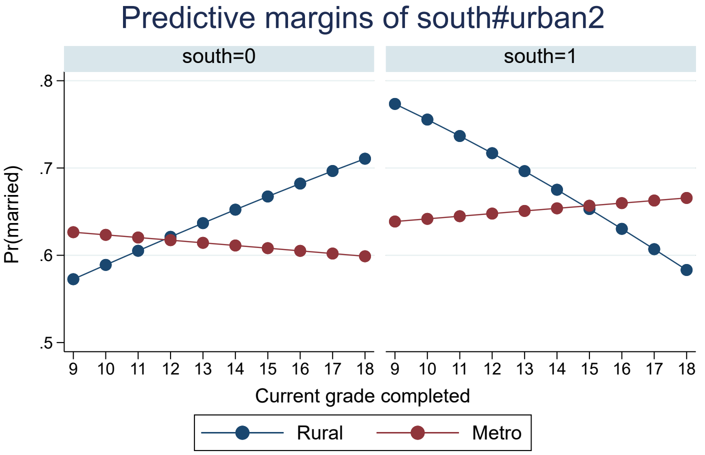
```
. margins south#urban2, at(grade=(9(1)18))
```

```
marginsplot, noci
```



This example shows the graph created by the `marginsplot` command, graphing the predicted margins computed by the `margins` command. The `noci` option is used to suppress the confidence intervals. The following examples customize the graph, focusing on issues related to the by-dimension. *Uses nlsw.dta*

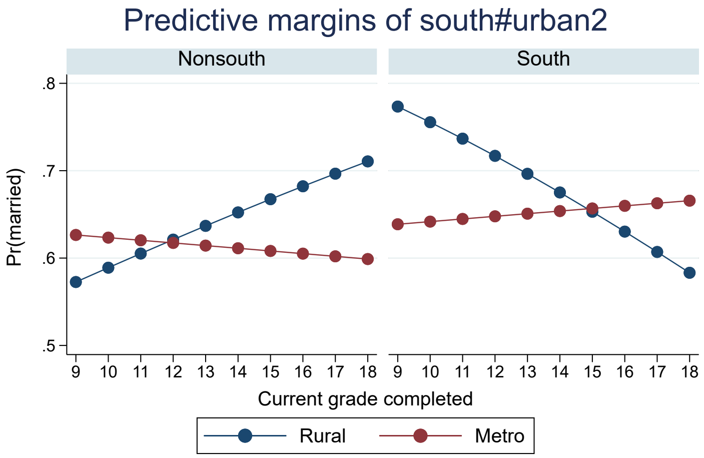
```
marginsplot, noci bydimension(south)
```



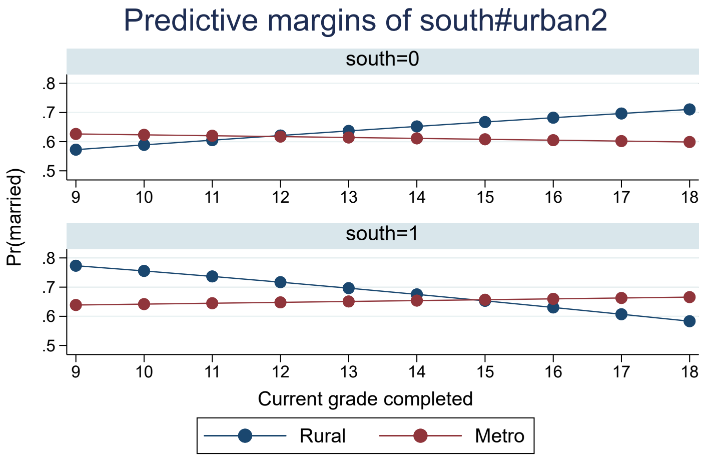
The `bydimension()` option is used to specify that separate graphs be created based on whether one lives in the South. *Uses nlsw.dta*

```
marginsplot, noci bydimension(south, label("Nonsouth"
"South"))
```

The `label()` suboption can be used to control the labeling of each of the graphs. You can control the labeling of the by-dimension by using the `bydimension()` option in the same manner that we controlled the plot dimension by using the `plotdimension()` option. *Uses nlsw.dta*



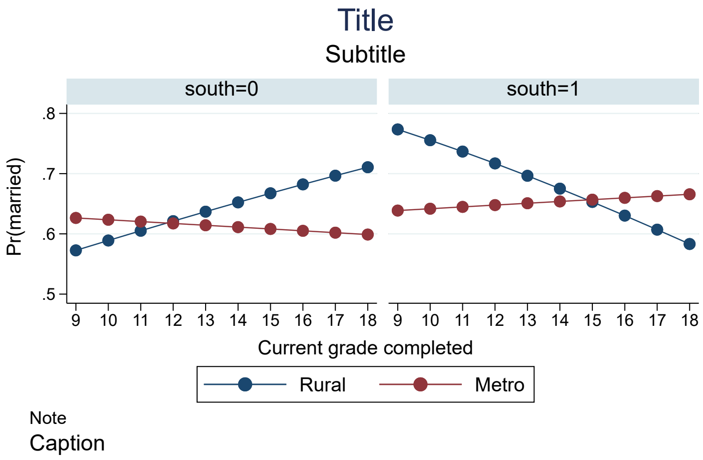
```
marginsplot, noci bydimension(south) byopts(cols(1) ixaxes)
```



The `byopts()` option allows you to specify suboptions that control the way the separate graphs are combined together. In this example, the `cols(1)` and `ixaxes` suboptions are specified to display the graphs in one column, each with its own x axis. You can see Options : By (section 8.8) for additional suboptions that you could supply within the `byopts()` option. *Uses nlsw.dta*

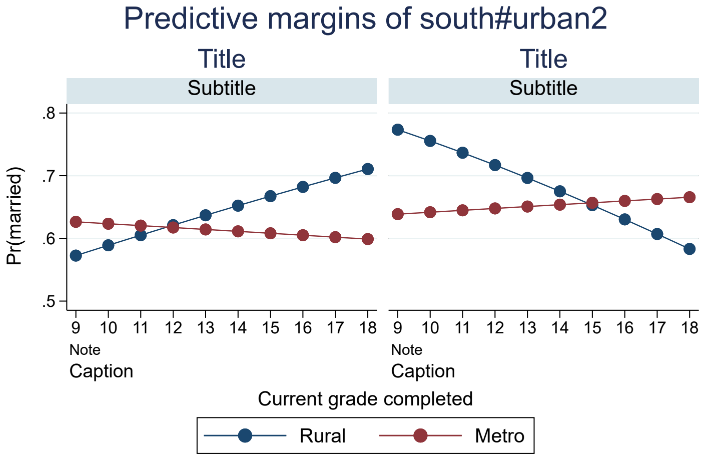
```
marginsplot, noci bydimension(south) byopts(title(Title)
subtitle(Subtitle) note(Note) caption(Caption))
```

The `byopts()` option can be used to control the overall title, subtitle, note, and caption for the graph. By placing such options within the `byopts()` option, these options impact the overall title, subtitle, note, and caption for the graph. Contrast this with the next example. *Uses nlsw.dta*



```
marginsplot, noci bydimension(south) title(Title)
subtitle(Subtitle) note(Note) caption(Caption)
```

By placing these options outside the `byopts()` option, these options control the title, subtitle, note, and caption for each of the graphs. This is generally not the desired result. *Uses nlsw.dta*



Let's now consider one last example, illustrating the use of `margins` and `marginsplot` after a multiple logistic regression model. The example below models the urbanization of a place where one lives ($1 = \text{Rural}$, $2 = \text{Suburb}$, $3 = \text{Urban}$) as a function of educational attainment (last grade completed). The `mlogit` command models `urban3` as a multinomial outcome as a function of education.

```
. use nlsw
. mlogit urban3 grade
```

Those with more education were less likely to live in a rural location ($\text{RRR} = 0.88, p < 0.01$), while living in an urban location was unrelated to educational level ($p = 0.78$).

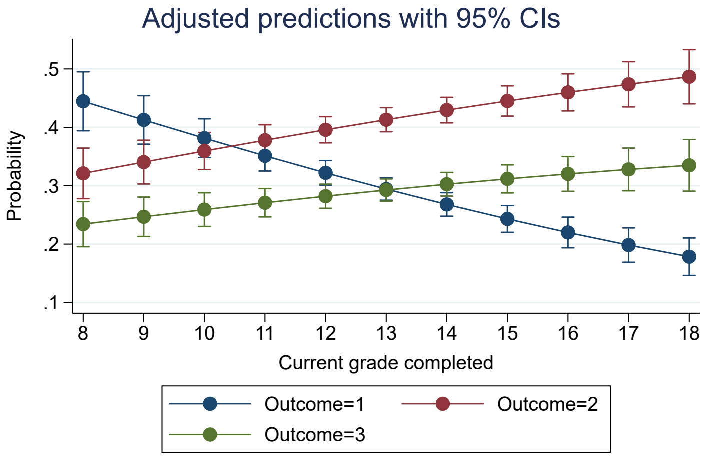
This result might be easier to understand if we could visualize the predictive margins as a function of education. First, we use the `margins` command to compute the predictive margin of each outcome by education, ranging from 8 to 18 years (in one-year increments).

```
. margins, at(grade=(8(1)18))
```

Then, we can use the `marginsplot` command to visualize the output of the `margins` command.

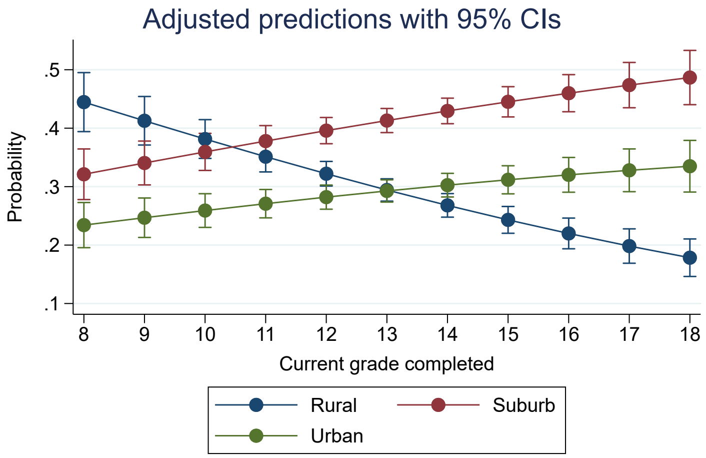
```
marginsplot
```

This example shows the graph created by the `marginsplot` command, graphing the predicted margins computed by the `margins` command. In the next example, we will make this graph clearer by adding labels associated with each outcome. *Uses nlsw.dta*

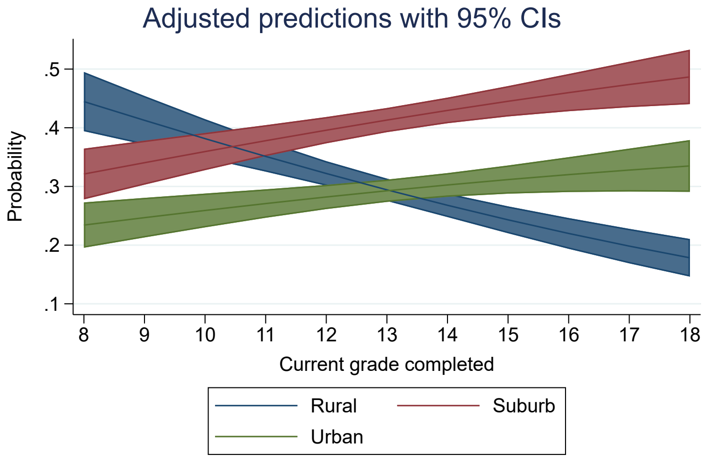


```
marginsplot, plotdim( , labels("Rural" "Suburb" "Urban"))
```

To make the interpretation of the graph easier, let's add the `plotdim( , labels("Rural" "Suburb" "Urban"))` option, which labels the different plotted outcomes. Now it is easier to see that the probability of living in a rural location decreases with increased education. It might be easier to see this association if we displayed the confidence intervals using confidence regions via shaded areas. *Uses nlsw.dta*



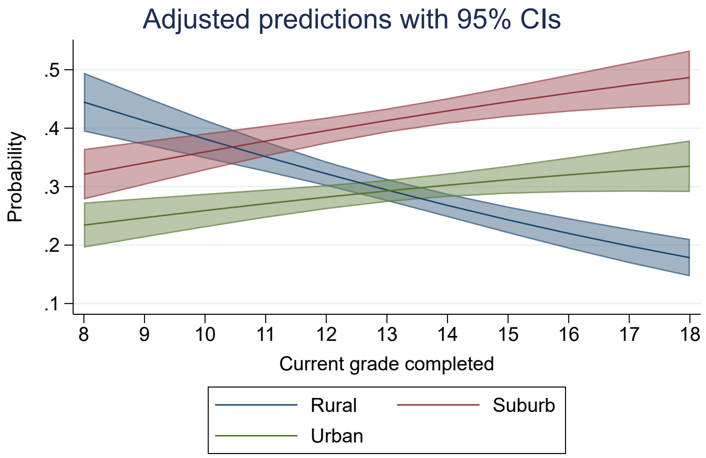
```
marginsplot, plotdim( , labels("Rural" "Suburb" "Urban"))
recastci(rarea) recast(line)
```



When we add the `recastci(rarea)` option, the confidence region is graphed using a shaded area plot. We also include the `recast(line)` option to graph the predictive margins using a line plot (omitting the markers). Unfortunately, the confidence regions are fully opaque, so the overlapping areas conceal the overlapped values. *Uses nlsw.dta*

```
marginsplot, plotdim( , labels("Rural" "Suburb" "Urban"))
recastci(rarea) recast(line) ciopts(color(%50))
```

When we add the `ciopts(color(%50))` option, the confidence region is displayed using 50% opacity. In the areas where the confidence regions overlap, we see the color of both regions. For example, where the confidence region for Suburb overlaps Rural, the reduced opacity allows the navy region to show underneath the maroon region. *Uses nlsw.dta*



This section has illustrated some of the ways that you can customize graphs created by the `marginsplot` command. As we have seen, the `marginsplot` command supports standard graph options [as described in Standard options (section 9)]. The `marginsplot` command also supports options that you would apply to a `twoway` graph [as described in Options (section 8)]. In addition, this section illustrated options that are specific to the `marginsplot` command, such as the `plotopts()`, `ciopts()`, and `plotdimension()` options. For more information, you can see `help marginsplot`.

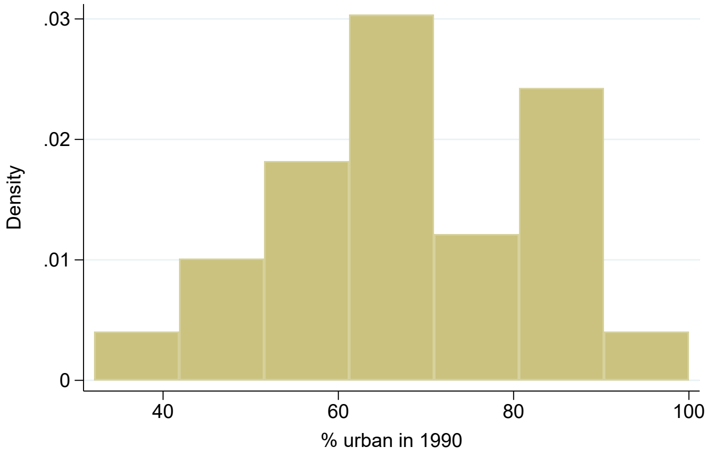
11.4 Saving, redisplaying, and combining graphs

This section shows how to save, redisplay, and combine Stata graphs.

The section begins by showing how to save graphs and use saved graphs. We can save graphs in one of two forms: *live* graphs or *as-is* graphs. We can edit a live graph by using the Graph Editor, and we can use and redisplay it with a different scheme. By contrast, an as-is graph can be displayed only exactly as it was saved—it can neither be edited nor be displayed with a different scheme. Such as-is graphs are generally smaller and will appear on another person's computer exactly as it looked when it was saved. Now let's look at a few examples.

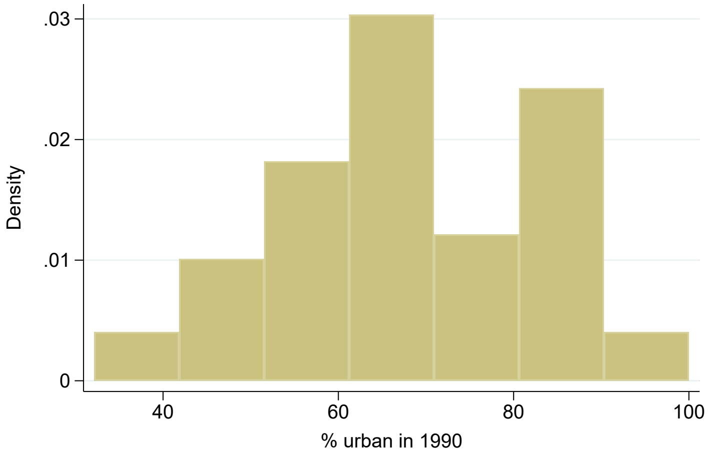
```
twoway histogram urban
```

Let's start by creating a histogram. Suppose that we liked this graph so much that we wanted to save it for either sending to someone else (who owns Stata) or displaying it at a later time. *Uses allstates.dta*

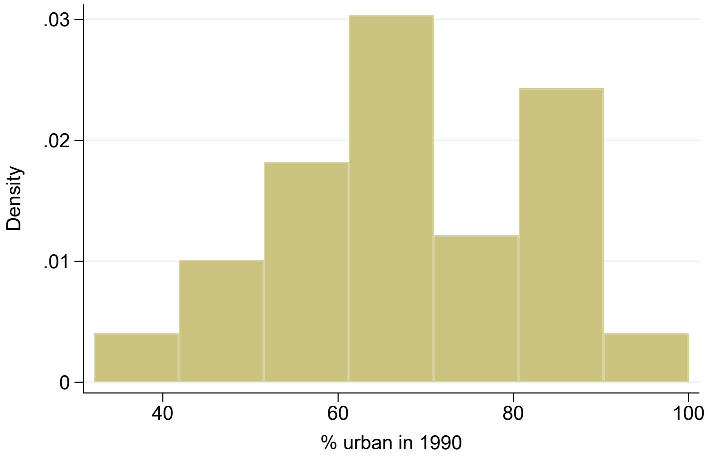


```
graph save hist1
```

The `graph save` command saves the currently displayed graph as a Stata `.gph` file. We save this graph in the current directory under the name `hist1.gph`. We will assume that in the following examples all graphs are stored in the current directory, but we can precede the filename with a directory name and store it wherever we want. This graph is stored as a live graph. If you added the `asis` option, the graph would have been stored as an as-is graph. *Uses allstates.dta*



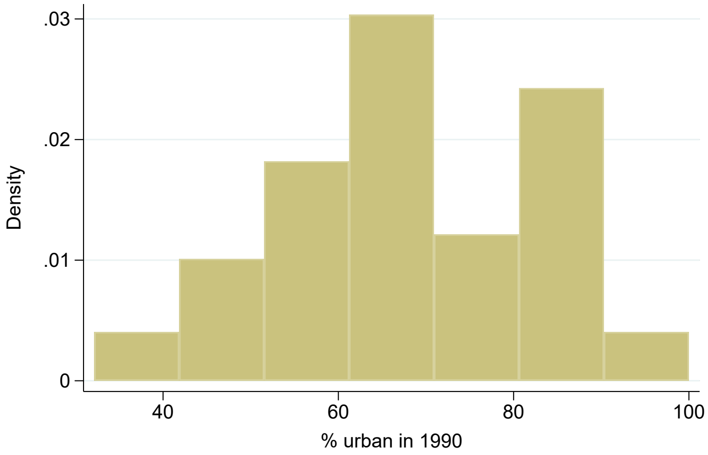
```
twoway histogram urban, saving(hist2, asis)
```



Most, if not all, Stata graph commands allow us to use the `saving()` option to save the graph as a Stata `.gph` file. This option allows us to create and save the graph in one step. Here we add the `asis` option to request the graph be stored as an as-is graph instead of a live graph. If `hist2.gph` existed, we would add the `replace` (next to the `asis` option), to overwrite the existing `hist2.gph` file. *Uses allstates.dta*

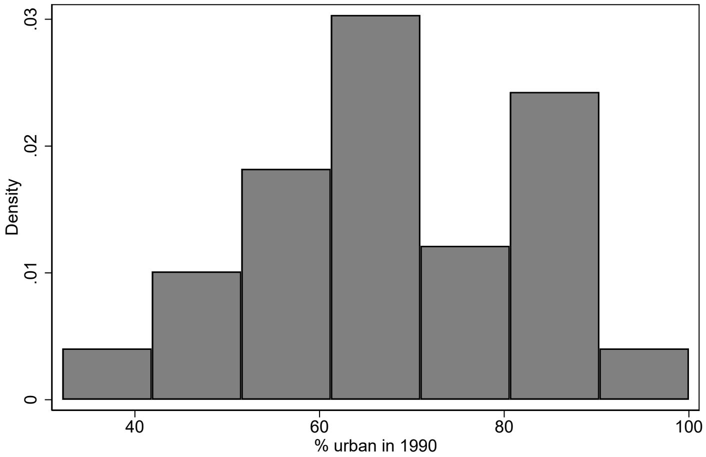
```
graph use hist1.gph
```

At a later time (including after quitting and restarting Stata), we can view a saved graph with the `graph use` command. Here we redisplay `hist1.gph`. If `hist1.gph` had been stored in a different directory, you would have to precede it with the directory where it was saved or use the `cd` command to change to that directory. *Uses allstates.dta*



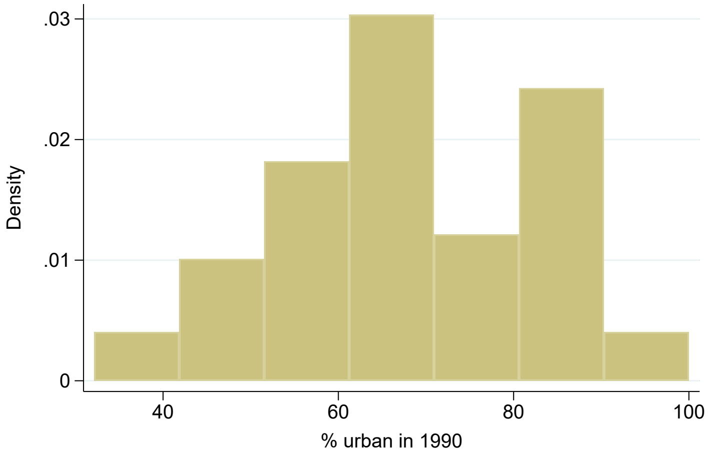
```
graph use hist1.gph, scheme(s1mono)
```

Because `hist1.gph` is a live graph, we can add the `scheme()` option to view the same graph using a different scheme. Here we view the last graph but use the `s1mono` scheme. *Uses allstates.dta*

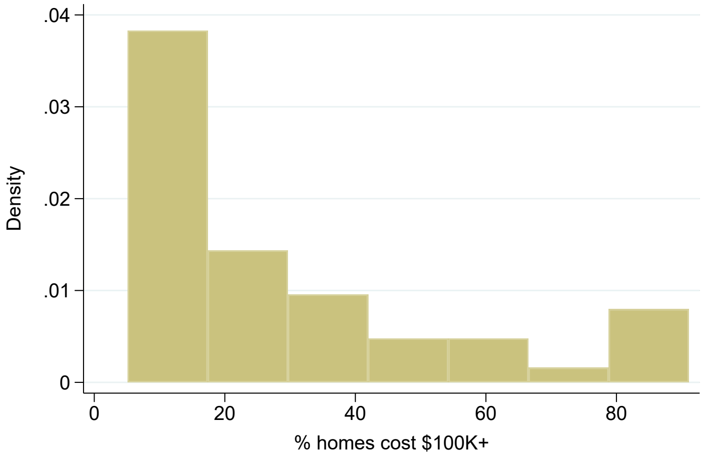


```
graph use hist2.gph
```

By contrast, `hist2.gph` is an as-is graph. We can display this graph with the `graph use` command, but we cannot display it with a different scheme. *Uses allstates.dta*



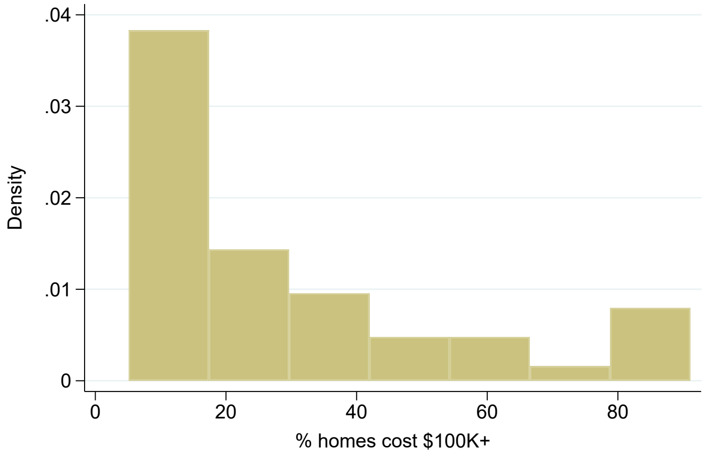
```
twoway histogram propval100, name(hist2)
```



The `name()` option is much like the `saving()` option, except that the graph is saved in memory instead of on disk. We can then view the graph later within the same Stata session, but once we quit Stata, the graph in memory is erased. *Uses allstates.dta*

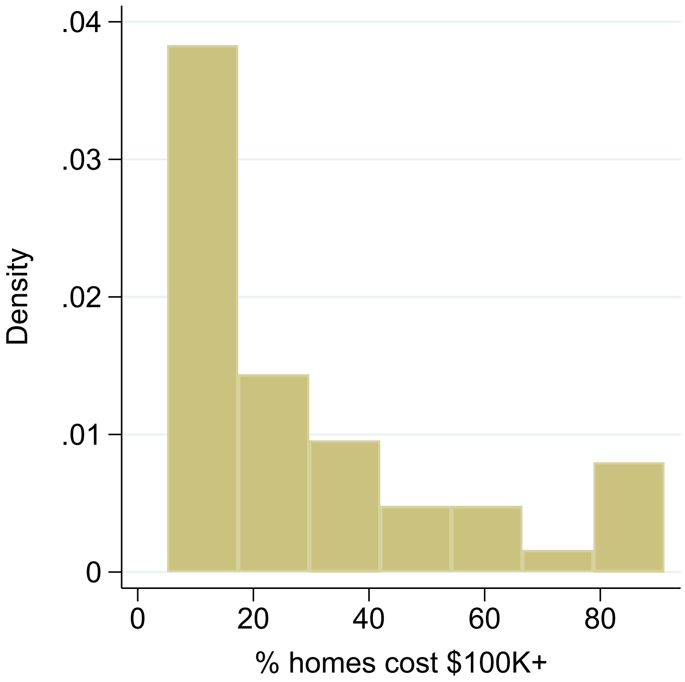
```
graph display hist2
```

The `graph display` command is similar to the `graph use` command, except that it redisplays graphs saved in memory. Here we redisplay the graph we created with the `name(hist2)` option. *Uses allstates.dta*



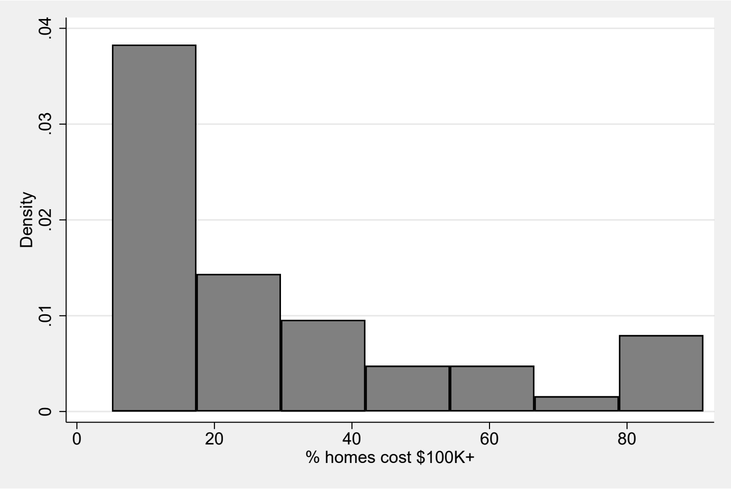
```
graph display hist2, xsize(2in) ysize(2in)
```

The `graph display` command allows us to use the `xsize()` and `ysize()` options to change the size and aspect ratio of the graph. Here we redisplay the graph we named `hist2` and make the graph 2 inches tall by 2 inches wide. *Uses allstates.dta*



```
graph display hist2, scheme(s2mono)
```

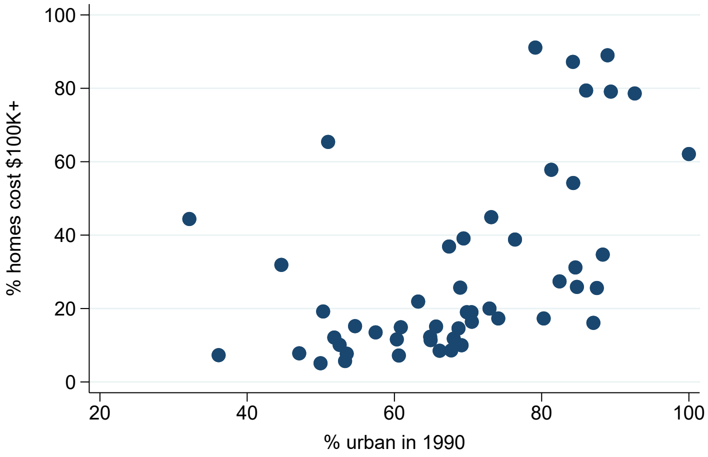
We can also use the `scheme()` option to view the same graph using a different scheme. Here we view the previous graph but with the `s2mono` scheme. *Uses allstates.dta*



Let's now look at some examples that illustrate how to combine graphs once we have created and saved them. First, we will see how to show two scatterplots side by side rather than overlaying them.

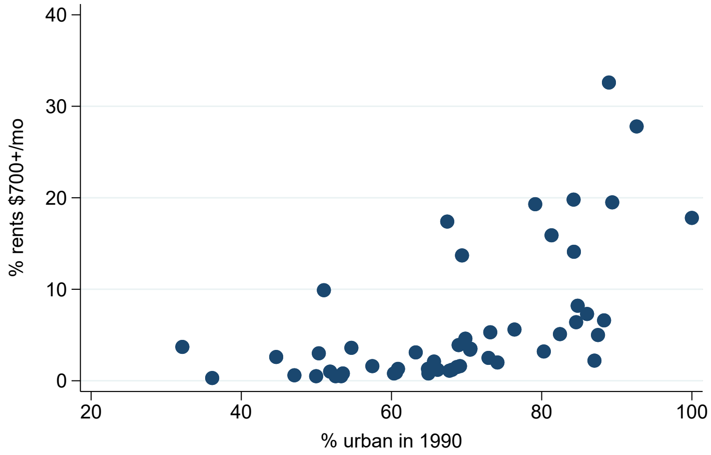
```
twoway scatter propval100 urban, name(scat1)
```

Using the `name(scat1)` option saves this scatterplot in memory with the name `scat1`. *Uses allstates.dta*

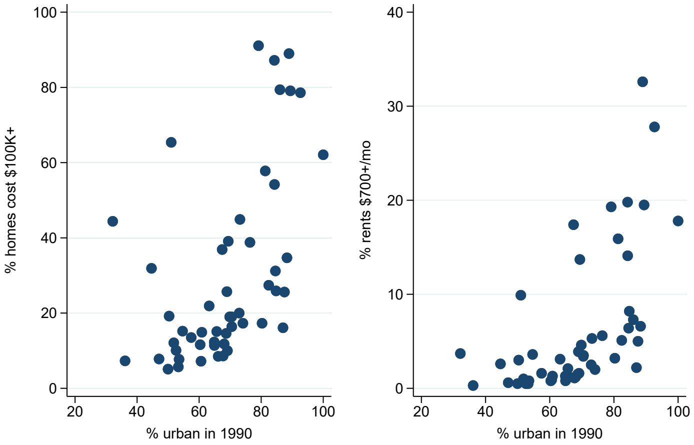


```
twoway scatter rent700 urban, name(scat2)
```

We save this second scatterplot with the name `scat2`. *Uses allstates.dta*



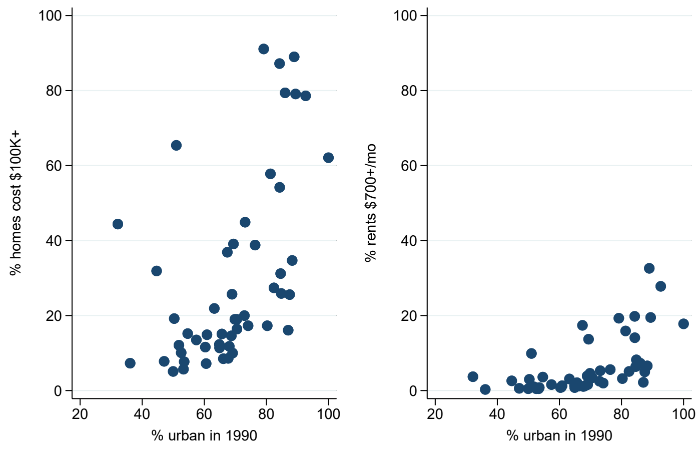
```
graph combine scat1 scat2
```



By using the `graph combine` command, we can see these two scatterplots side by side. In a sense, the y axis is on a different scale for these two graphs because they are different variables. However, in another sense, the scale for the two y axes is the same because they are both measured as percentages. *Uses allstates.dta*

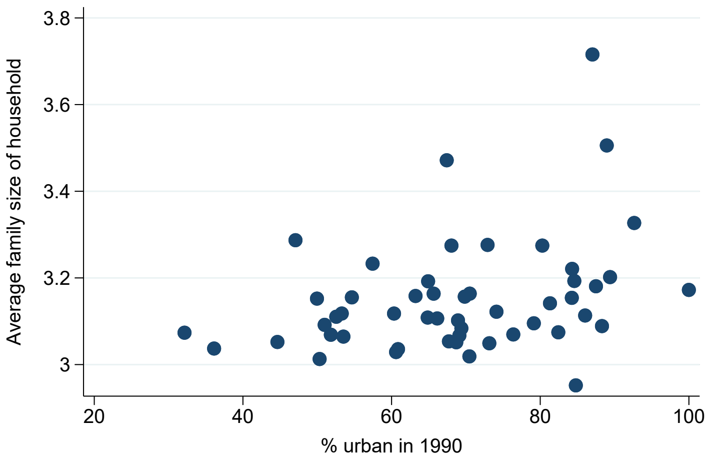
```
graph combine scat1 scat2, ycommon
```

This graph is the same as the last one, except that the y axes are placed on a common scale by using the `ycommon` option. This makes it easy to compare the two y variables by forcing them to be on the same metric. The `ycommon` option does not work when the graphs have been made by using different kinds of commands, for example, `graph bar` and `graph box`. *Uses allstates.dta*



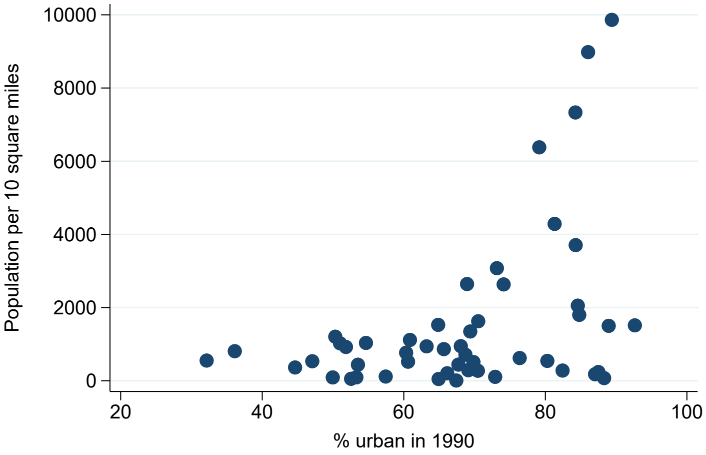
```
twoway scatter famsize urban, name(scat3)
```

Let's make two more graphs. This graph is named `scat3`. *Uses allstates.dta*



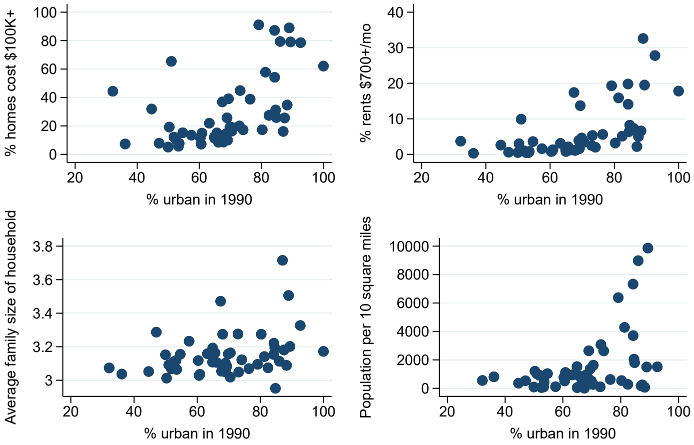
```
twoway scatter popden urban, name(scat4)
```

This example is named `scat4`. The next example will illustrate combining the four graphs `scat1`, `scat2`, `scat3`, and `scat4` into one graph. *Uses allstates.dta*



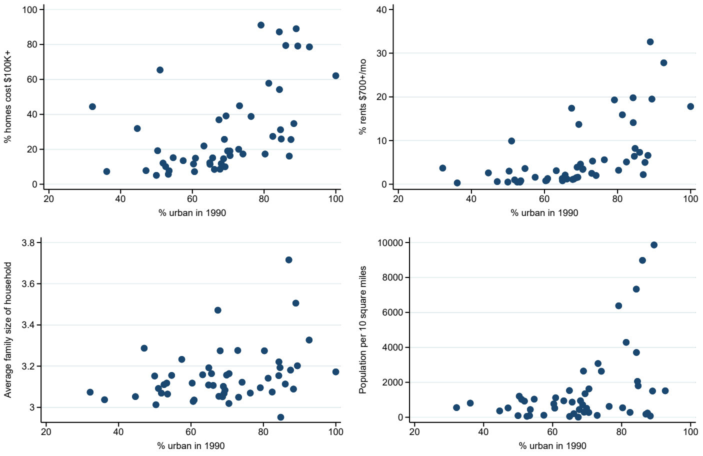
```
graph combine scat1 scat2 scat3 scat4
```

Note the default sizing of the text and markers in this combined graph. The next graph shows an alternate sizing we can select. *Uses allstates.dta*



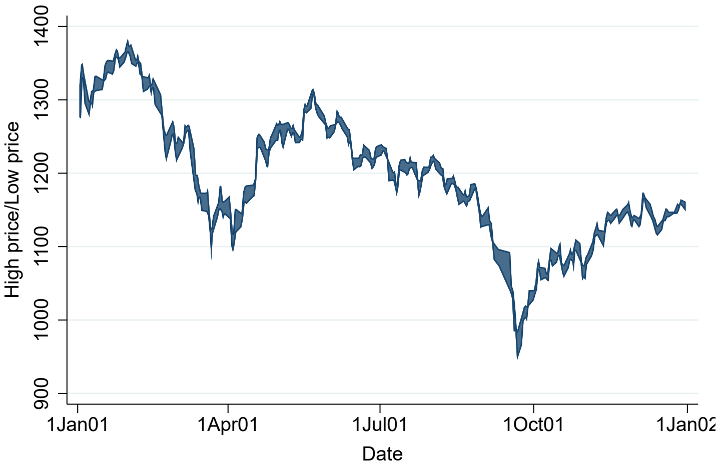
```
graph combine scat1 scat2 scat3 scat4, altshrink
```

Compare this graph with the previous graph. Note how the `altshrink` option reduces the size of the text and markers. Sometimes this option might be useful when combining graphs. *Uses allstates.dta*



Here are more detailed examples showing how you can combine graphs and the options to use in creating the graphs. The next set of examples uses the `sp2001ts` data file.

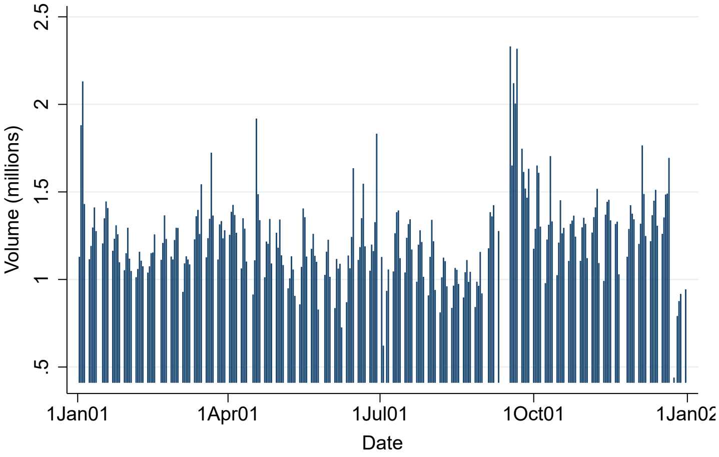
```
twoway rarea high low date, name(hilo) ylabel(,
angle(vertical))
```



We make a graph showing the high and low closing price of the S&P 500 for 2001 and save this graph in memory, naming it `hilo`. Note the addition of the `ylabel(, angle(vertical))` option, which will be useful when we combine the graphs in a later step. *Uses sp2001ts.dta*

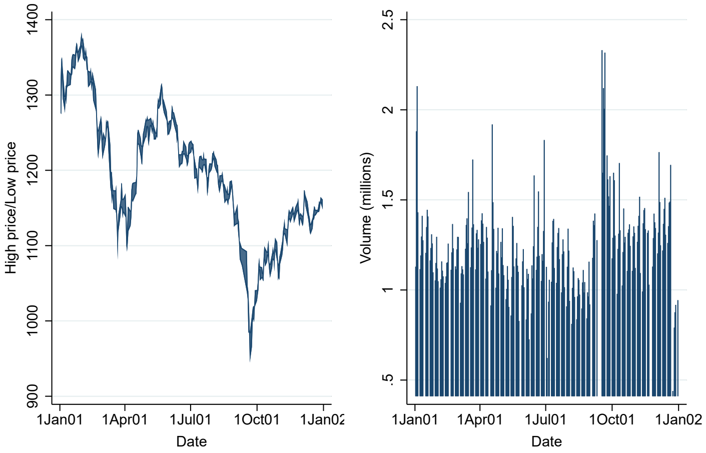
```
twoway spike volmil date, name(vol) ylabel(, angle(vertical))
```

We can make another graph that shows the volume (millions of shares sold per day) for 2001 and save this graph in memory, naming it `vol`. For this graph, also note the addition of the `ylabel(, angle(vertical))` option. This option will be useful when we combine this graph with the prior graph in a later step. *Uses sp2001ts.dta*



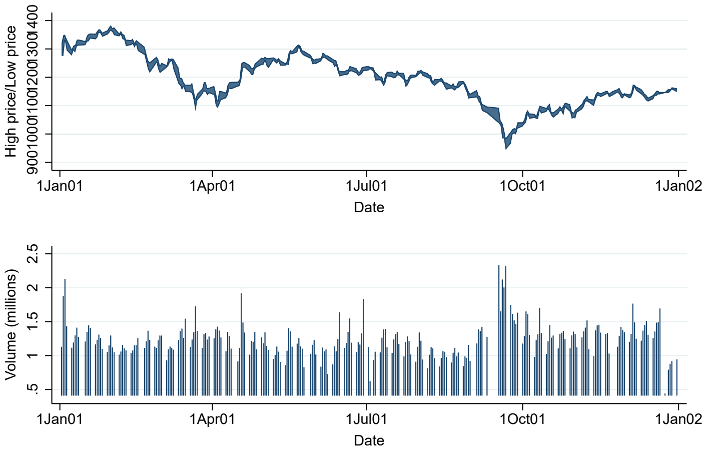
```
graph combine hilo vol
```

We can now use the `graph combine` command to combine these two graphs into one graph. The graphs are displayed as one row, but say that we would like to display them in one column. *Uses sp2001ts.dta*



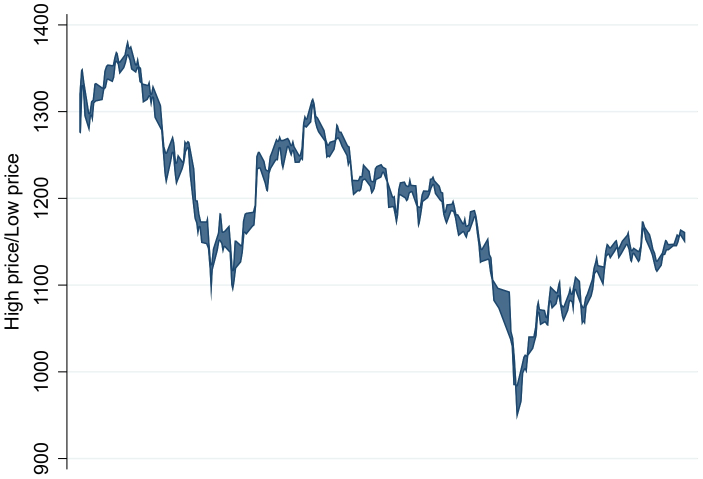
```
graph combine hilo vol, cols(1)
```

By using the `cols(1)` option, we can display the price above the volume. However, because the x axes of these two graphs are scaled the same, we could save space by removing the x-axis scale from the top graph. The two graphs align better because each used the `ylabel(, angle(vertical))` option. Had we omitted this option, the y axes of the graphs would not have been aligned. *Uses sp2001ts.dta*



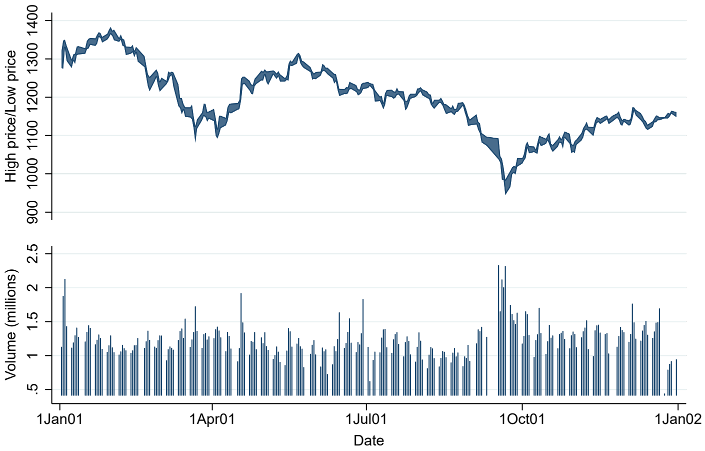
```
twoway rarea high low date, xscale(off) name(hilo, replace)
ylabel(, angle(vertical))
```

Here we use the `xscale(off)` option to suppress the display of the x axis, including the space that would be allocated for the labels. We name this graph `hilo` again, but we must use the `replace` option to replace the existing graph named `hilo`. *Uses sp2001ts.dta*



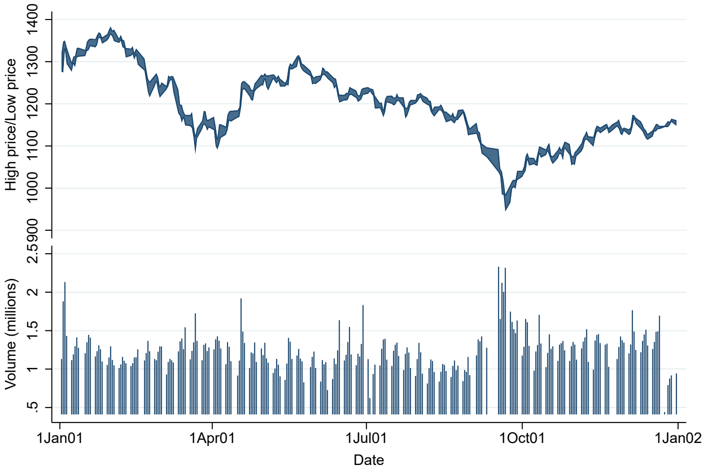
```
graph combine hilo vol, cols(1)
```

We combine these two graphs; however, we might want to push the graphs a bit closer together. *Uses sp2001ts.dta*



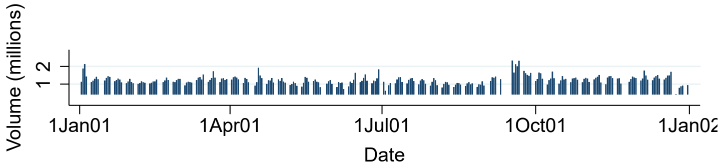
```
graph combine hilo vol, cols(1) imargin(b=1 t=1)
```

Here we use the `imargin(b=1 t=1)` option to make the margin at the top and bottom of the graphs to be small before combining them. However, we might want the lower graph of volume to be smaller. *Uses sp2001ts.dta*

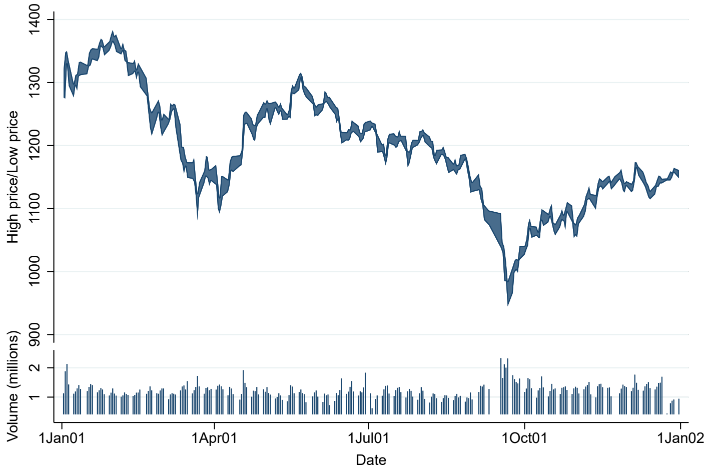


```
twoway spike volmil date, ylabel(1 2) fysize(25) name(vol,
replace) ylabel(, angle(vertical))
```

Using the `fysize()` (force y size) option makes the graph 25% of its normal size. We use this instead of `ysize()` because the `graph combine` command does not respect the `ysize() or xsize()` options. For aesthetics, we also reduce the number of labels. We save this graph in memory, replacing the existing graph named `vol`. *Uses sp2001ts.dta*



```
graph combine hilo vol, cols(1) imargin(b=1 t=1)
```



We combine these graphs again, and the combined graph looks pretty good. We might further tinker with the graph, changing the `xtitle()` for the volume graph to be shorter or modifying the `xlabel()` for the volume graph. Also, note that including the `ylabel(, angle(vertical))` option when creating each graph improved the alignment of the combined graph. *Uses sp2001ts.dta*

11.5 Exporting graphs

There are many different reasons that we might export a Stata graph into another format. We might want to post the graph onto our website, or we might want to email it to a friend (who does not have Stata), or we might be submitting it with a manuscript for publication. This section will explore different ways to export files into different formats. For more information on exporting graphs, type `help graph export`.

There are several questions you might ask yourself while you are in the process of creating the graph. Some questions are related to the use and selection of a graph scheme, including the following:

1. What scheme should I use for my graph? Do I want to use one of the schemes built into Stata, or do I want to use one of the community-contributed schemes? [See Standard options : Schemes (section 9.2) for information about different schemes.]
2. Can the graph be in *color*, or does it need to be in black and white? If the graph will be in *black and white*, you may want to use a scheme specifically designed for black and white. [See Standard options : Schemes (section 9.2) for more details.]
3. Are there formatting requirements, such as the presence or absence of grid lines or background color? Some publishers prefer graphs that do not have shading in the *graph region* (that is, the area outside of the plot where the titles and axes are displayed). To avoid such shading, you can select a graph scheme that omits such shading. Or you can manually specify options to omit such shading [see Standard options : Graph regions (section 9.4)].
4. Do the elements of the graph need to be sized in exact units (like points, inches, or centimeters)? Or would you prefer to use relative sizing? Either way, you can specify the sizing of graphical elements as shown in Standard options : Sizing graphs (section 9.3).
5. Should the graph include a figure title?
6. What size should the graph be? The required size could be specified in terms of the height and width, or it could be specified in terms of a minimum resolution.

7. Are there limitations on the file format you will export to? For example, I believe that `.ps` or `.eps` files do not currently support opacity, so you may want to avoid specifying colors using opacity when exporting to those file formats. Also, I believe that `.ps` and `.eps` do not currently support Unicode. If you are exporting to those formats, you may want to avoid using Unicode.
8. Is the specified file format supported by Stata on my computer? (You can consult `help graph export` for a list of the file formats Stata can create and any restrictions due to particular operating systems.) For example, as of the writing of this book, `.emf` files can be exported only on Stata for Windows for the current version of Stata. Further, at this time, only the macOS version of Stata can create `.gif` files.

Once you have created your graph, you can then export it into the file format of your choice. My recommendation is that you use the `graph export` command to export files.[1] Before you export the graph, you might need to specify some technical options. See below.

1. `.pdf`: If you are exporting to a `.pdf` file, there are no options to specify. The size of your `.pdf` file will be controlled by the `xsize()` and `ysize()` options specified when the graph was created. I illustrate this in the “snippets” below.
2. `.jpg`: When using `graph export`, you can specify the height and width in pixels as well as the quality. See `help jpg_options` for more information.
3. `.png`: When using `graph export`, you can specify the height and width in pixels. See `help png_options` for more information.
4. `.tif`: When using `graph export`, you can specify the height and width in pixels. See `help tif_options` for more information.
5. `.gif`: When using `graph export`, you can specify the height and width in pixels. Note that `.gif` is available only with Stata for Mac. See `help gif_options` for more information.
6. `.eps` files: When you use `graph export`, there are a few technical options you may need to specify, although the default options may suffice in many cases. See `help eps_options` for more details.
7. `.ps` files: When you use `graph export`, there are a few technical questions involved in exporting such files, although the default options

may suffice in many cases. See `help ps_options` for more details.

8. `.svg` files: There are many options you can specify when exporting `.svg` files; see `help svg_options`.

I suggest that you place your `graph export` commands in same do-file that you are using to create the graph. Thus, if you make any changes to the command that creates the graph, you just execute the do-file, which will re-create the graphs and export them all in one step. The following will show some snippets illustrating this process, creating the graph and then using the `graph export` command to export the graph into a new file.[2]

In *Snippet #1* below, I create a graph that is 4-inches wide and 3-inches tall and export it to a `.pdf` file named `fig1.pdf`. Note that the size was specified on the `scatter` command.[3] When exporting the `.pdf` file, the `graph export` command sizes the graph according to the size when the graph was created (in this case, 4-inches wide and 3-inches tall because I specified the `xsize(4in)` and `ysize(3in)` options on the `scatter` command). Also note that I began the snippet with the `set scheme s2color` command, specifying the scheme to be used for the graphs unless I specify otherwise with the `scheme()` option.

```
. * Snippet #1
. set scheme s2color
. use allstates, clear
. scatter workers2 pcturban80, mcolor(red) title("This is the Title") ///
    xsize(4in) ysize(3in)
. graph export fig1.pdf
```

Suppose I am submitting a graph for publication and the publisher wants the file in `.eps` format, sized 4 inches by 4 inches, displayed in a landscape orientation, and using CMYK colors. This kind of graph is created using *Snippet #2* shown below. The `scatter` command is used to specify the graph size via the `xsize(4in)` and `ysize(4in)` options. The `graph export` command exports the graph as `fig1.eps` using landscape orientation and using CMYK colors. You can see `help eps_options` for more details on exporting `.eps` files.

```
. * Snippet #2
. set scheme s2color
. use allstates, clear
. scatter workers2 pcturban80, mcolor(red) title("This is the Title") ///
.   xsize(4in) ysize(4in)
. graph export fig1.eps, orientation(landscape) cmyk(on)
```

Imagine I am creating a figure that I want to post on my website. I decide that I want to save the figure as a `.png` file with a width of 800 pixels and a height of 600 pixels. This kind of figure is created using *Snippet #3*. Note that the `scatter` command does not specify the size of the graph. This is because the graph resolution is specified via the `graph export` command. The `graph export` command below creates a figure named `fig1.png` that has a width of 800 pixels and a height of 600 pixels.

```
. * Snippet #3
. set scheme s2color
. use allstates, clear
. scatter workers2 pcturban80, mcolor(red) title("This is the Title")
. graph export fig1.png, width(800) height(600)
```

Now, imagine I want to export my scatterplot as two `.pdf` files, one that is 4-inches wide by 3-inches tall and a second one that is 48-inches wide by 36-inches tall. This is illustrated in *Snippet #4* below. I use the `scatter` command to create the graph (without sizing it). Then, I resize it to be 4-inches wide and 3-inches tall[4] and export that graph as `fig1_4by3.pdf`; then, I resize the graph to be 48-inches wide and 36-inches tall and export that graph as `fig1_48by36.pdf`. This snippet is useful for saving the same graph using different sizes.

```
. * Snippet 4
. set scheme s2color
. use allstates, clear
. scatter workers2 pcturban80, mcolor(red) title("This is the Title")
. * 4 by 3 version
. graph display, xsize(4in) ysize(3in)
. graph export fig1_4by3.pdf
. * 48 by 36 version
. graph display, xsize(48in) ysize(36in)
. graph export fig1_48by36.pdf
```

The example shown in *Snippet #5* below is much like *Snippet #4*, except that the small graph requires omission of the title and the large graph requires a title. This snippet is useful when you need to create two graphs of differing sizes that are nearly identical.

```
. * Snippet 5
. set scheme s2color
. use allstates, clear
. * 4 by 3 version omits the title.
. scatter workers2 pcturban80, mcolor(red)
. graph display, xsize(4in) ysize(3in)
. graph export fig1_4by3.pdf
. * 48 by 36 version
. scatter workers2 pcturban80, mcolor(red) title("This is the Title")
. graph display, xsize(48in) ysize(36in)
. graph export fig1_48by36.pdf
```

Exporting graphs is often a critical step in the creation of a graph. The way that you export the graph can have a large impact on the final look of the graph. For more details about exporting graphs, see `help graph export`.

---

1. You can export graphs via *point-and-click* methods—for example, you can right-click on the graph and then use the pop-up menu to help you save the graph, select the file format, and specify the filename. This method might have fewer export options as compared with the `graph export` command. For more information, see `help graph export`, notably the section titled *Exporting the graph displayed in a Graph window*.

2. Please note that these snippets omit commands that are recommended within a do-file such as `log using` to open a log file, `log close` to close a log file, and `version` to specify the Stata version number used when the do-file was created.

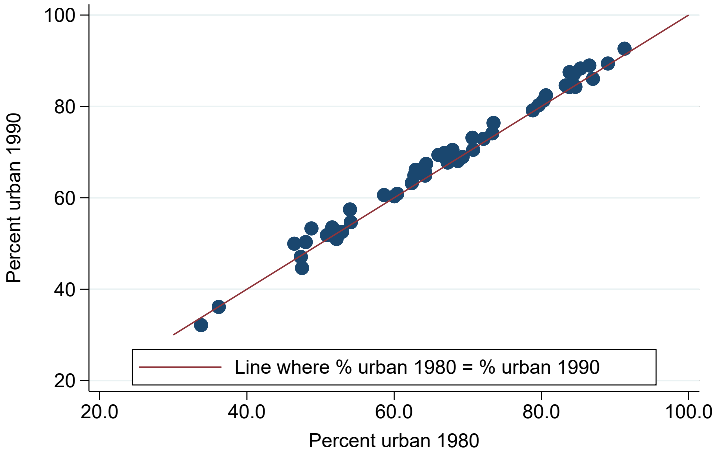
3. Also note that the `scatter` command has only two customizations, specifying a title and the color of the markers. Normally, a graph would have more customizations, but these examples are focused less on the customizations and more on the process of creating and exporting graphs.

4. I use the `graph display` command combined with the `xsize()` and `ysize()` options to resize the graph currently in memory.

11.6 More examples: Putting it all together

Most examples in this book have focused on the impact of one option or a few options, using datasets that required no manipulation before making the graph. In reality, many graphs use multiple options, and some require prior data management. This section addresses this issue by showing some examples that combine many options and require some data manipulation before making the graph.

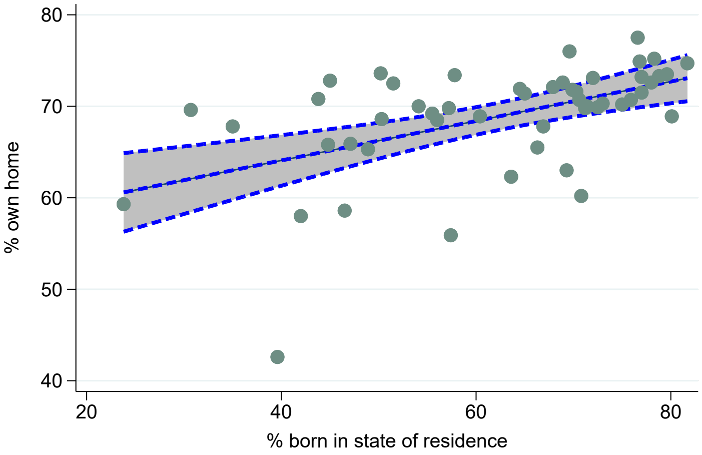
```
twoway (scatter urban pcturban80) (function y=x, range(30
100)), xtitle(Percent urban 1980) ytitle(Percent urban 1990)
legend(order(2 "Line where % urban 1980 = % urban 1990")
pos(6) ring(0))
```



This graph shows the percentage of population living in an urban area of a state in 1990 against that of 1980. If there had been no changes from 1980 to 1990, the values would fall along a 45-degree line, where the value of y equals the value of x. Overlaying `(function y=x)`, we can see any discrepancies from 1980 to 1990. The `range(30 100)` option makes the line span from 30 to 100 on the x axis. *Uses allstates.dta*

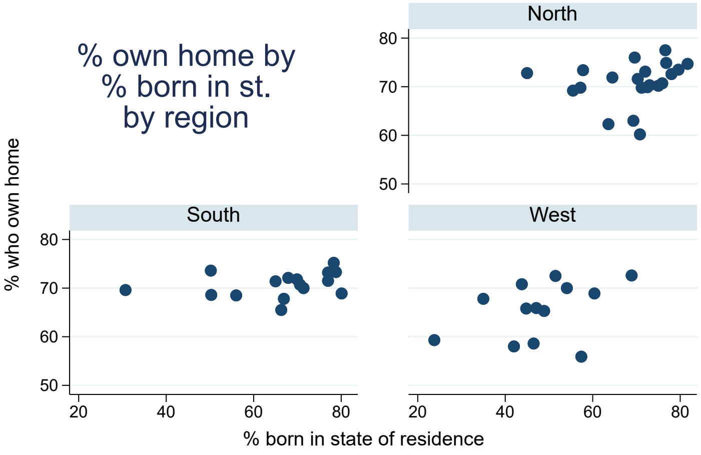
```
twoway (lfitci ownhome borninstate) (lfitci ownhome
borninstate, ciplot(rline) lcolor(blue) lwidth(thick)
lpattern(dash)) (scatter ownhome borninstate), legend(off)
ytitle("% own home")
```

This example shows how we can make a scatterplot, a regression line, and a confidence interval for the fit shown as an area. We also add a thick, blue, dashed line showing the upper and lower confidence limits. The first `lfitci` makes the fit line and area; the second `lfitci` makes a thick, blue, dashed outline for the area; and `scatter` overlays the scatterplot. *Uses allstates.dta*



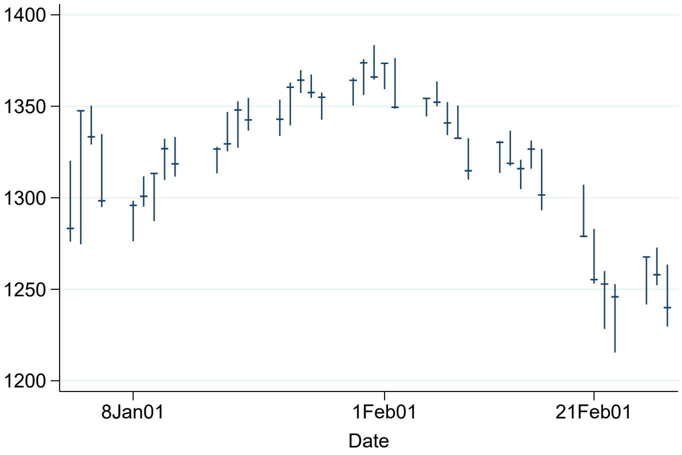
```
twoway scatter ownhome borninstate, by(nsw, hole(1) title("%
own home by" "% born in st." "by region", pos(11) ring(0)
width(65) height(35) justification(center) alignment(middle))
note(""))
```

The `hole(1)` option leaves the first position empty when creating the graphs, and the title is placed there using `pos(11)` and `ring(0)`. We use `width()` and `height()` to adjust the size of the textbox and `justification()` and `alignment()` to center the textbox horizontally and vertically. The `note("")` option suppresses the note in the bottom corner of the graph. *Uses allstates.dta*



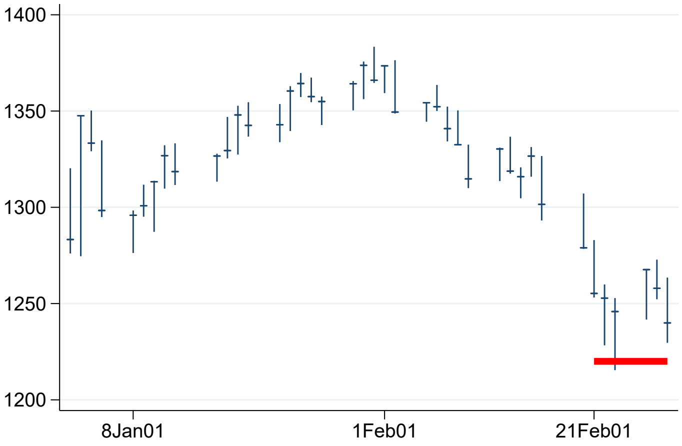
```
twoway (rspike high low date) (rcap close close date,
msize(medsmall)), tlabel(08jan2001 01feb2001 21feb2001)
legend(off) scheme(vg_samec)
```

Before making this high/low/close graph, we first type `tsset date, daily` to tell Stata that `date` should be treated as a date in the `tlabel()` option. The `rcap` command uses `close` for both the high and the low values, making the tick line for the closing price, and the `legend(off)` option suppresses the legend. Using the `vg_samec` scheme makes the spikes and caps the same color. *Uses spjanfeb2001.dta*



```
twoway (rspike high low date) (rcap close close date,
msize(medsmall)) (scatteri 1220 15027 1220 15034,
recast(line) clwid(vthick) clcol(red)), tlabel(08jan2001
01feb2001 21feb2001) legend(off) scheme(vg_samec)
```

This example is the same as above, except that this one uses `scatteri()` to draw a support-level line. Two y x pairs are given after the `scatteri`, and the `recast(line)` option draws them as a line instead of two points. The x values were calculated beforehand by using `display d(21feb2001)` and `display d(28feb2001)` to compute the elapsed date values. *Uses spjanfeb2001.dta*



The rest of the examples in this section involve some data management before creating the graph. The next few examples use the `allstates` data file. First, run a regression command:

```
. use allstates

. regress ownhome propval100 workers2 urban
```

Then issue the

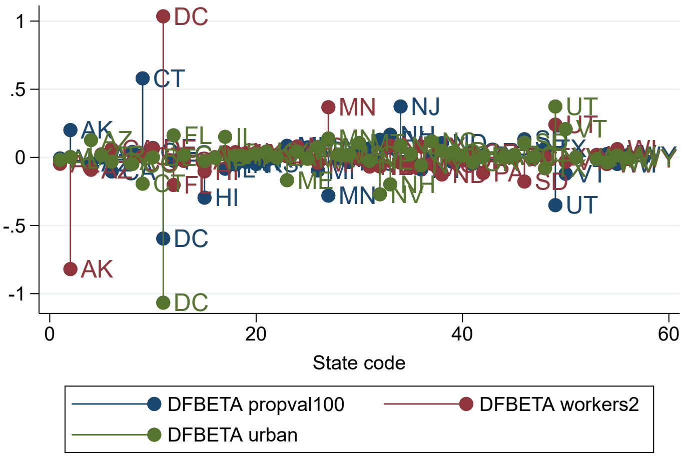
```
. dfbeta
```

command, creating DFBETAs for each predictor: `DFpropval100`, `DFworkers2`, and `DFurban`, which are used in the following graph. We also generated a sequential ID variable with the following command:

```
. generate id = _n
```

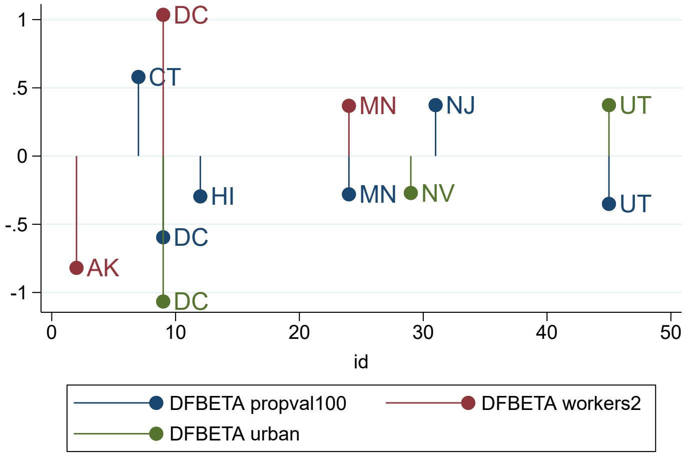
```
twoway dropline _dfbeta_1 _dfbeta_2 _dfbeta_3 statefips,
mlabel(stateab stateab stateab)
```

Here we show each DFBETA as a `dropline` plot. We add the `mlabel()` option to label each point with the state abbreviation. *Uses allstates.dta*



```
twoway (dropline _dfbeta_1 id if abs(_dfbeta_1)>.25,
mlabel(stateab)) (dropline _dfbeta_2 id if
abs(_dfbeta_2)>.25, mlabel(stateab)) (dropline _dfbeta_3 id
if abs(_dfbeta_3)>.25, mlabel(stateab))
```

This example is similar to the one above but simplifies the graph by showing only the points where the DFBETA exceeds 0.25. Note that we have taken the example from above and converted it into three overlaid `dropline` plots, each of which has an `if` condition. *Uses allstates.dta*

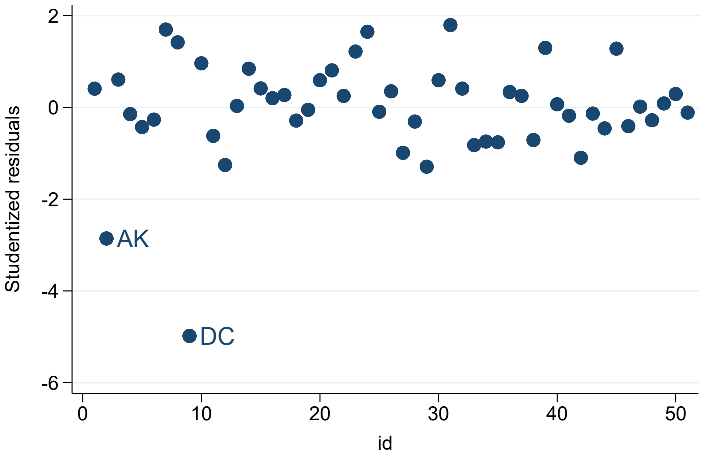


Before making the next graph, we must issue three `predict` commands to generate variables that contain the Cook's distance, the Studentized residual, and the leverage from the previous regression command:

```
. predict cd, cook
. predict rs, rstudent
. predict l, leverage
```

We are now ready to create the next graph.

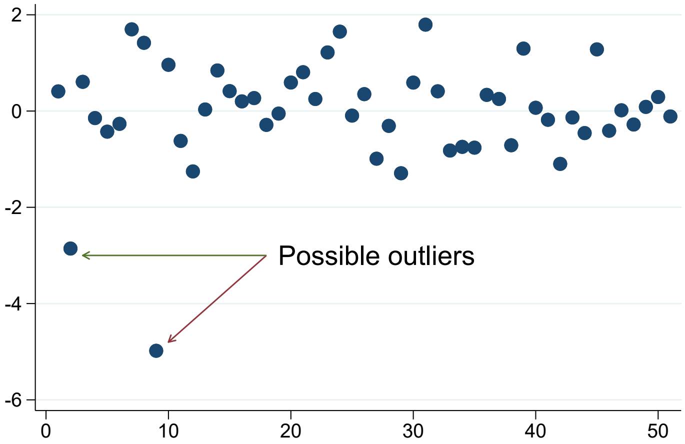
```
twoway (scatter rs id) (scatter rs id if abs(rs) > 2,
mlabel(stateab)), legend(off) scheme(vg_samec)
```



This graph uses `scatter rs id` to make an index plot of the Studentized residuals. It also overlays a second `scatter` command with an `if` condition showing only Studentized residuals that have an absolute value exceeding 2 and showing the labels for those observations. Using the `vg_samec` scheme makes the markers the same for both `scatter` commands. *Uses allstates.dta*

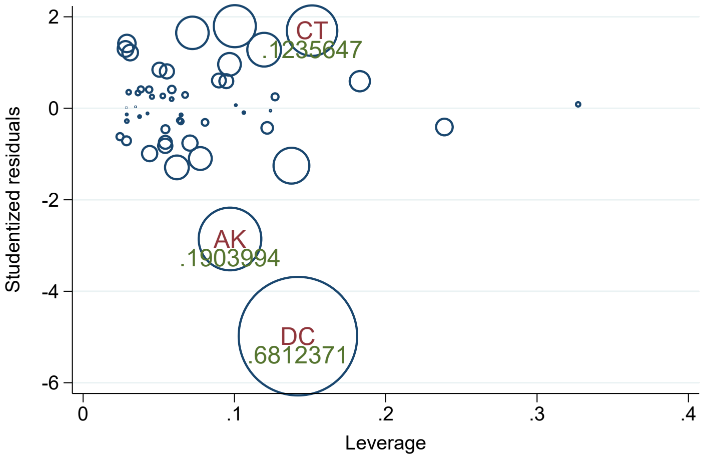
```
twoway (scatter rs id, text( -3 27 "Possible outliers",
size(vlarge))) (pcarrowi -3 18 -4.8 10) (pcarrowi -3 18 -3
3), legend(off)
```

This graph is similar to the one above but uses the `text()` option to add text to the graph. It also uses two `pcarrowi` commands to draw a line ending with an arrow from the text “Possible outliers” to the markers for those points. The y and x coordinates are given for the starting and ending positions. *Uses allstates.dta*



```
twoway (scatter rs l [aw=cd], msymbol(Oh)) (scatter rs l if
cd > .1, msymbol(i) mlabel(stateab) mlabpos(0)) (scatter rs l
if cd > .1, msymbol(i) mlabel(cd) mlabpos(6)), legend(off)
```

This graph shows the leverage-versus-Studentized residuals, weighting the symbols by Cook’s D (`cd`). We overlay it with a scatterplot showing the marker labels if `cd` exceeds 0.1, with the `cd` value placed below. *Uses allstates.dta*

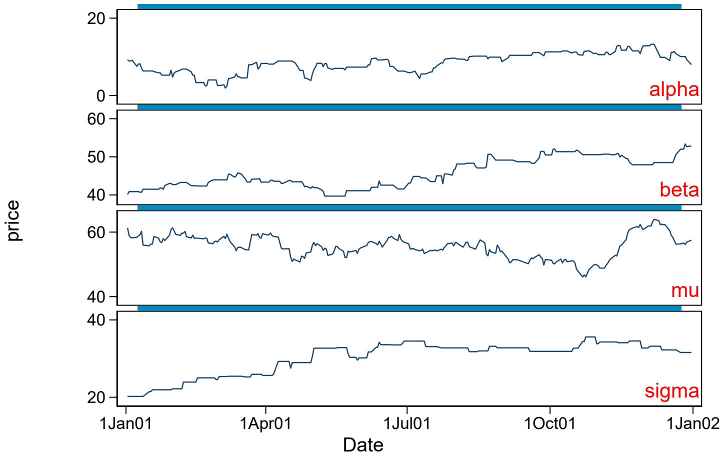


Say that we have a data file called `comp2001ts` that contains variables representing the stock prices of four hypothetical companies: `pricealpha`, `pricebeta`, `pricemu`, and `pricesigma`, as well as a variable `date`. To compare the performance of these companies, let's make a line plot for each company and stack them. We can do this by using `twoway tsline` with the `by(company)` option, but we first need to reshape the data into a `long` format. We do so with the following commands:

```
. use comp2001ts, clear

. reshape long price, i(date) j(compname) string
```

We now have variables `price` and `company` and can graph the prices by company.

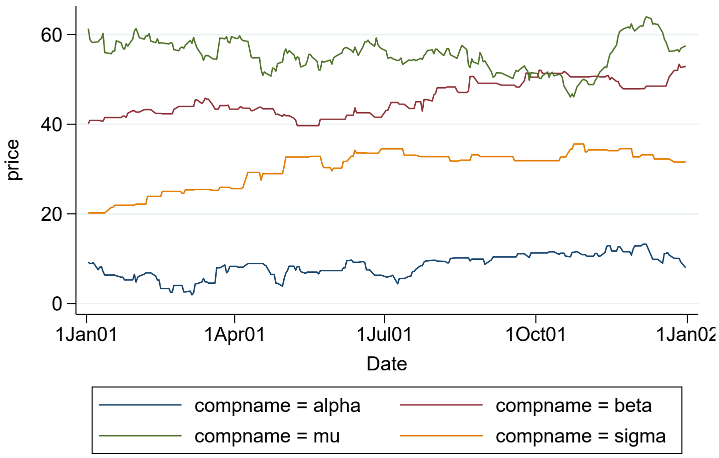
```
twoway tsline price, by(compname, cols(1) yrescale note("")
compact) ylabel(#2, nogrid) subtitle(, pos(5) ring(0)
nobexpand nobox color(red)) title(" ", box width(130)
height(.001) bcolor(ebblue))
```



We graph `price` for the different companies with the `by()` option. The `cols(1)` option puts the graphs in one column. `yrescale` and `ylabel(#2)` allow the y axes to be scaled independently and labeled with about 2 values. The `subtitle()` option puts the name of the company in the bottom right corner of each graph. The `title()` option combined with the `compact` option creates a blue border between the graphs. *Uses comp2001ts.dta*

```
xtline price, i(compname) t(date) overlay
```

We can also use the `xtline` command to graph these same companies in one graph. The `i()` option specifies the variable that uniquely identifies an observation (for example, the company), and the `t()` option provides the variable that represents time (plotted on the x axis). The `overlay` option places all the companies in the same graph. *Uses comp2001ts.dta*



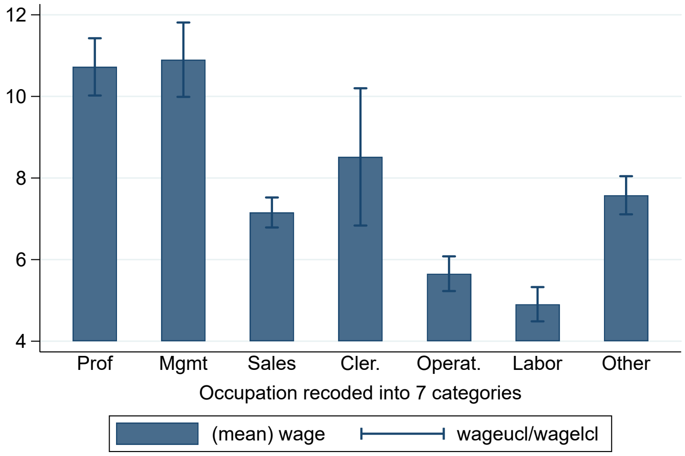
For the next graph, we want to create a bar chart that shows the mean of wages by occupation with error bars showing a 95% confidence interval for each mean. First, we collapse the data across the levels of occupation, creating the mean, standard deviation, and count. Next we create the variables `wageucl` and `wagelcl`, which are the upper and lower confidence limits, as shown below.

```
. use nlsw
. collapse (mean) mwage=wage (sd) sdwage=wage (count) nwage=wage, by(occ7)
. generate wageucl = mwage + invttail(nwage,0.025)*sdwage/sqrt(nwage)

. generate wagelcl = mwage - invttail(nwage,0.025)*sdwage/sqrt(nwage)
```

We are now ready to graph the data.

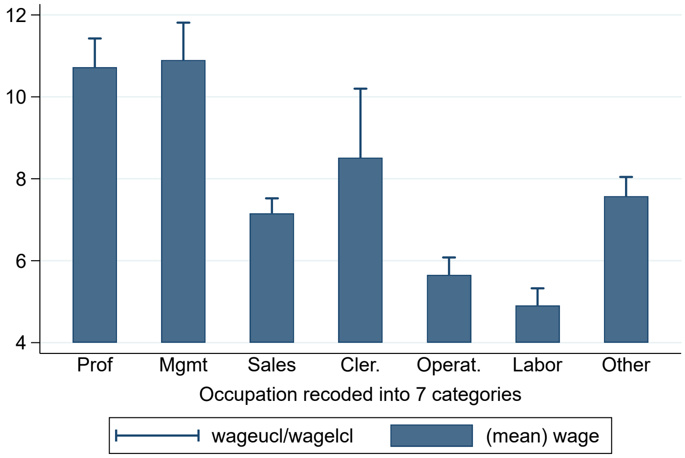
```
twoway (bar mwage occ7, barwidth(.5)) (rcap wageucl wagelcl
occ7, blwid(medthick) lcolor(navy) msize(large)),
xlabel(1(1)7, valuelabel noticks) xscale(range(.5 7.5))
```

This bar chart is overlaid with a range plot showing the upper and lower confidence limits. The `xlabel()` option labels the values from 1 to 7, incrementing by 1. The `valuelabel` option indicates that the value labels for `occ7` will be used to label the x axis. The `xscale()` option adds a margin to the outer bars, and the `barwidth()` option creates the gap between the bars. *Uses nlsw.dta*



```
twoway (rcap wageucl wagelcl occ7, lwidth(medthick)
msize(large)) (bar mwage occ7, barwidth(.5) bcolor(navy)),
xlabel(1(1)7, valuelabel noticks) xscale(range(.5 7.5))
```

This graph is similar to the previous one, except that we have reversed the order of the commands, placing the `rcap` command first, followed by the `bar` command. As a result, only the top half of the error bar is shown. As in the previous example, the `xlabel()` option determines the labels on the x axis. *Uses nlsw.dta*

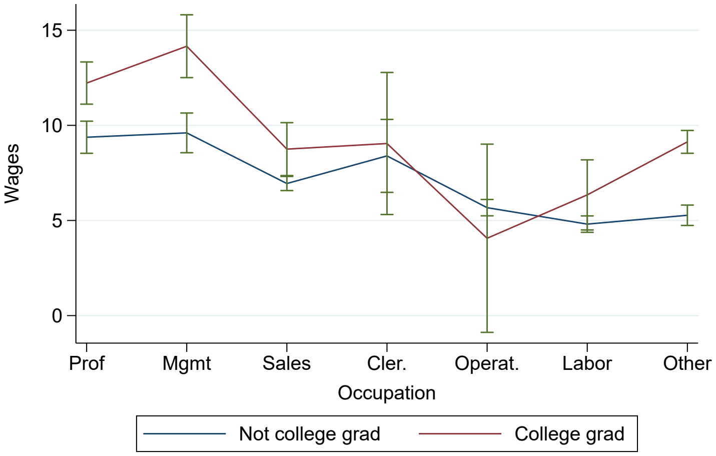


Suppose that we wanted to show the mean wages with confidence intervals broken down by occupation and whether one graduated college. We can use the `collapse` command to create the mean, standard deviation, and count by the levels of `occ7` and `collgrad`. We can then create the upper and lower confidence limits. Finally, the `separate` command makes separate variables of `mwage` based on whether one graduated college, creating `mwage0` (wages for noncollege grad) and `mwage1` (wages for college grad). These commands are shown below, followed by the command to create the graph.

```
. use nlsw, clear
. collapse (mean) mwage=wage (sd) sdwage=wage
(count) nwage=wage, by(occ7 collgrad)
. generate wageucl = mwage + invttail(nwage,0.025)*sdwage/sqrt(nwage)
. generate wagelcl = mwage - invttail(nwage,0.025)*sdwage/sqrt(nwage)
. separate mwage, by(collgrad)
```

```
twoway (line mwage0 mwage1 occ7) (rcap wageucl wagelcl occ7),
xlabel( 1(1)7, valuelabel) xtitle(Occupation) ytitle(Wages)
legend(order(1 "Not college grad" 2 "College grad"))
```

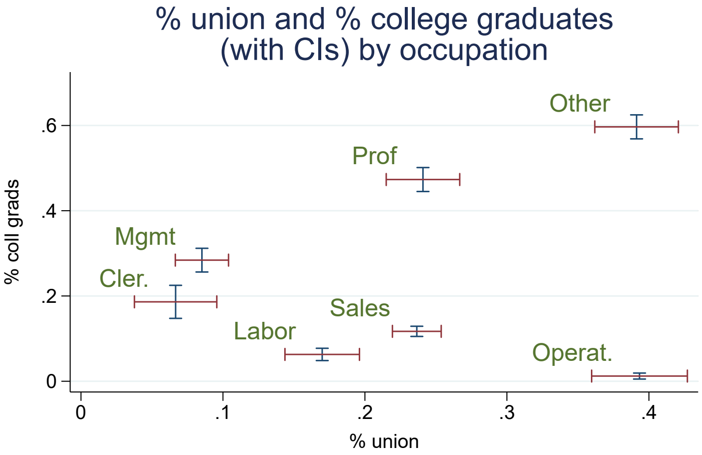
Here we make a line graph showing the mean wages for the noncollege graduates, `mwage0`, and the college graduates, `mwage1`, by occupation. We overlay that with a range plot showing the confidence interval. The `xlabel()` option labels the x axis with value labels, and the `legend()` option labels the legend. *Uses nlsw.dta*



This next graph shows a type of scatterplot of the mean and confidence interval for `union` and `collgrad` for each level of `occ7`. To do this, collapse the data file by `occ7` and use those summary statistics to compute the confidence intervals below, followed by the command to create the graph.

```
. use nlsw, clear

. collapse (mean) pct_un=un pct_coll=collgrad
(sd) sd_un=union sd_coll=collgrad
(count) ct_un=union ct_coll=collgrad, by(occ7)
. generate lci_un = pct_un - sd_un/sqrt(ct_un)
. generate uci_un = pct_un + sd_un/sqrt(ct_un)
. generate lci_coll = pct_coll - sd_coll/sqrt(ct_coll)

. generate uci_coll = pct_coll + sd_coll/sqrt(ct_coll)
```

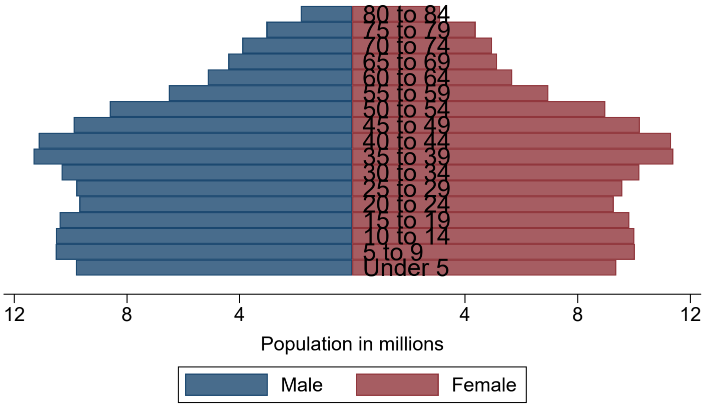
```
twoway (rcap lci_coll uci_coll pct_un) (rcap lci_un uci_un
pct_coll, hor) (sc pct_coll pct_un, msymbol(i) mlabel(occ7)
mlabpos(10) mlabgap(5)), ylabel(0(.2).7) xtitle(% union)
ytitle(% coll grads) legend(off) title("% union and % college
graduates" "(with CIs) by occupation")
```



The overlaid `rcap` commands show the confidence intervals for both `union` and `collgrad` for each occupation. The `scatter` command uses an invisible marker and labels each occupation at the ten o'clock position with a larger gap than normal. *Uses nlsw.dta*

This section concludes with a graph adapted from an example on the Stata web site. The graph combines numerous tricks, so rather than show it all at once, let's build it a piece at a time. Below is the ultimate graph we wish to create. It shows the population (in millions) for males and females in 17 different age groups, ranging from "Under 5" to "80–84". The blue bar represents the males, and the red bar represents the females.

```
graph display
```



This is the graph that we wish to create. For now, we simply use the `graph display` command to display the graph. This is shown using the `s2color` scheme *Uses pop2000mf.dta*

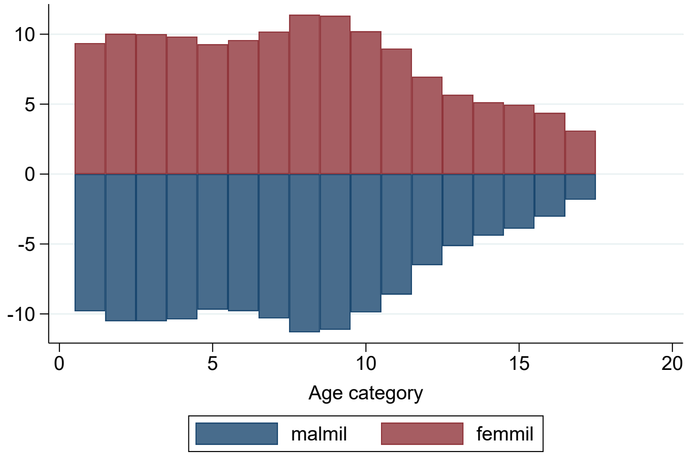
To build this graph, we first use the data file `pop2000mf`, which contains 17 observations corresponding to 17 age groups (for example, “Under 5”, “5–9”, “10–14”, and so forth). The variables `femtotal` and `maletotal` contain the number of females and males in each age group. After loading the data into Stata, we create `femmil`, which is the number of females per million, and `malmil`, which is the number of males per million, but this is made negative so that the male (blue) bar will be scaled in the negative direction. We also generate a variable called `zero`, which contains 0 for all observations.

```
. use pop2000mf, clear
. generate femmil = femtotal/1000000
. generate malmil = -maletotal/1000000
. generate zero = 0
```

We now take the first step in making this graph.

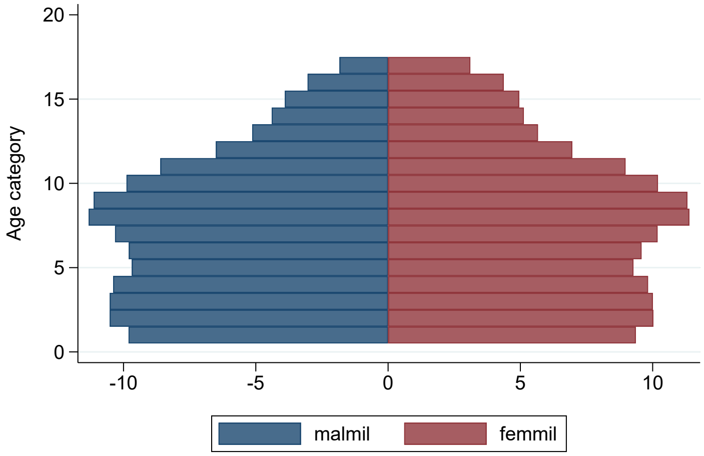
```
twoway (bar malmil agegrp) (bar femmil agegrp)
```

This is our first attempt to make this graph by overlaying the bar chart for the males with the bar chart for the females. The `agegrp` variable ranges from 1 to 17 and forms the x axis, but we can rotate this as shown in the next example. *Uses pop2000mf.dta*

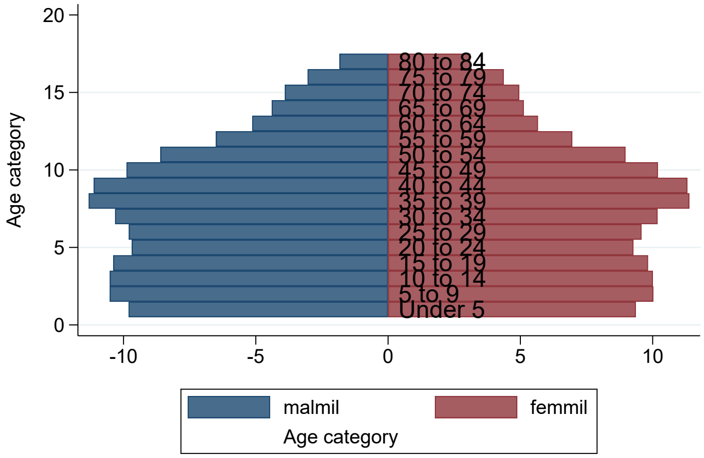


```
twoway (bar malmil agegrp, horizontal) (bar femmil agegrp,
horizontal)
```

Adding the `horizontal` option to each bar chart, we can see the graph taking shape. However, we would like the age categories to appear inside the red (female) bars. *Uses pop2000mf.dta*



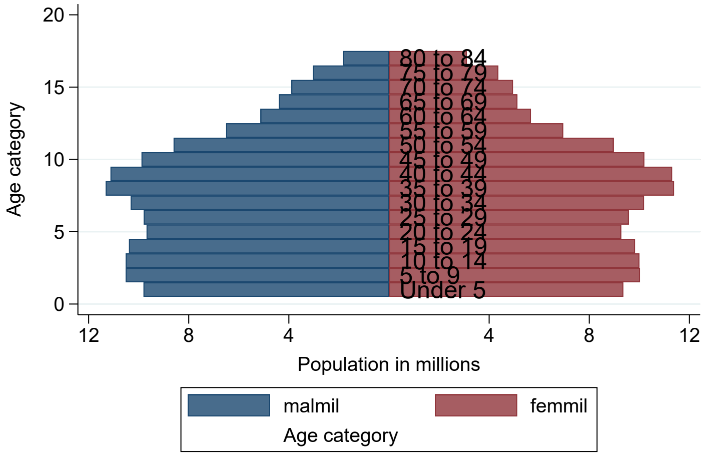
```
twoway (bar malmil agegrp, horizontal) (bar femmil agegrp,
horizontal) (scatter agegrp zero, msymbol(i) mlabel(agegrp)
mlabcolor(black))
```



This `scatter` command uses `agegrp` (ranging from 1 to 17) as the y value and `zero` (0) for the x value, leading to the stack of 17 observations. Using the `msymbol()` and `mlabel()` options suppresses the symbol but displays the name of the age group from the labeled value of `agegrp`. Next we will fix the label and title for the x axis. *Uses pop2000mf.dta*

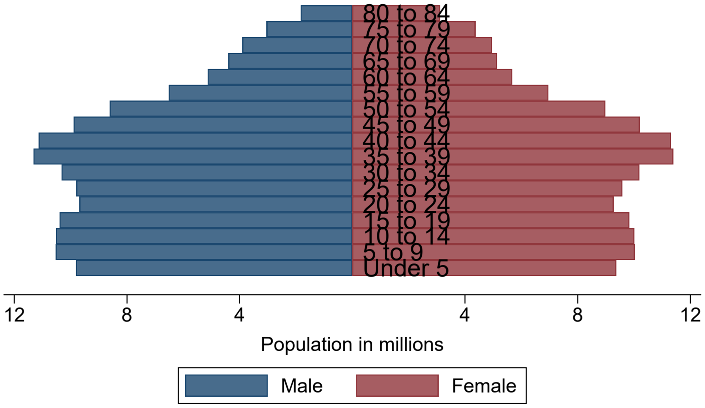
```
twoway (bar malmil agegrp, horizontal) (bar femmil agegrp,
horizontal) (scatter agegrp zero, msymbol(i) mlabel(agegrp)
mlabcolor(black)), xlabel(-12 "12" -8 "8" -4 "4" 4 8 12)
xtitle("Population in millions")
```

We use the `xlabel()` to change -12 to 12, -8 to 8, -4 to 4, and to label the positive side of the x axis as 4, 8, and 12. We also add a title for the x axis. Next let's fix the y axis and the legend. *Uses pop2000mf.dta*



```
twoway (bar malmil agegrp, horizontal) (bar femmil agegrp,
horizontal) (scatter agegrp zero, msymbol(i) mlabel(agegrp)
mlabcolor(black)), xlabel(-12 "12" -8 "8" -4 "4" 4 8 12)
xtitle("Population in millions") yscale(off) ylabel(, nogrid)
legend(order(1 "Male" 2 "Female"))
```

We suppress the display of the y axis by using the `yscale(off)` option and suppress the grid lines by using the `ylabel(, nogrid)` option. Finally, we use the `legend()` option to label the bars and suppress the display of the third symbol in the legend. *Uses pop2000mf.dta*



11.7 Common mistakes

This section discusses mistakes that are frequently made when creating Stata graphs.

Commas with graph options Graph options can accept their own options (sometimes referred to as *suboptions*); for example,

```
. twoway scatter propval100 popden rent700, xtitle("My Title", box)
```

The `xtitle()` option allows us to specify the x-axis title followed by a comma and a suboption that places a box around the x-axis title. If we had been content with the existing x-axis title and only wanted to add the box around the title, we could have issued this command:

```
. twoway scatter propval100 popden rent700, xtitle( , box)
```

Note the comma before the `box` option.

Now suppose that we are content with the existing legend but want to make the legend display in one column.

```
. twoway scatter propval100 popden rent700, xtitle( , box) legend(cols(1))
```

Based on the syntax from the `title()` option, we might have been tempted to type `legend( , cols(1))`, but that would have led to an error. Some options, like the `legend()` option, simply take a list of options with no comma permitted.

Using options in the wrong context Consider the example below. Our goal is to move the labels for the x axis from their default position at the bottom of the graph to the alternate position at the top of the graph.

```
. twoway scatter propval100 rent700, xlabel( , alternate)
```

This command executes, but it does not have the desired effect. Instead, it staggers the labels of the x axis, alternating between the upper and lower positions. In this context, the `alternate` option means something different than we had intended. We really wanted to specify the `xscale(alternate)` option:

```
. twoway scatter propval100 rent700, xscale(alternate)
```

This command moves the entire scale of the x axis to the alternate position and has the desired effect. Another mistake we might have made was to put the `alternate` option as an overall option. This command is shown below with the result:

```
. twoway scatter propval100 rent700, alternate
option alternate not allowed
r(198);
```

Here we are half right. There is an option `alternate`, but we have used it in the wrong context, yielding the syntax error. The option we are specifying may be right, but we just need to put it into the right context.

Options appear to have no effect When we add an option to a graph, we generally expect to see the effect of adding the option. However, sometimes adding an option has no effect. Consider this example:

```
. twoway scatter propval100 rent700, mlabpos(12)
```

This command executes, but nothing changes as a result of including the `mlabpos(12)` option, which would change the position of the marker labels to the twelve o'clock position. There are no marker labels in the graph, so adding this option has no effect. We would have to use the `mlabel()` option to add marker labels before we would see the effect of this option.

Consider another example, which is a bit more subtle. We would like to make the line (periphery) of the marker thick. When we run the following command, we do not see any effect from adding the `mlwidth(thick)` option:

```
. twoway scatter propval100 rent700, mlwidth(thick)
```

The reason for this is that the marker has a `line` color and a `fill` color, and by default, they are the same color, so it is impossible to see the effect of changing the thickness of the line around the marker. However, if we make the `line` and `fill` colors different, as in the following example, we can see the effect of the `mlwidth()` option:

```
. twoway scatter propval100 rent700, mlwidth(thick)
         mlcolor(black) mfcolor(gs13)
```

Options when using by() Using the `by()` option changes the meaning of some options. Consider the following example:

```
. twoway scatter propval100 rent700, by(north) title(My title)
```

We might think that the `title()` option will provide a title for the entire graph, as it would when the `by()` option is not included. However, each graph will have "My title" as the title; the graph as a whole will not. To provide a title for the whole graph, we would specify the command this way:

```
. twoway scatter propval100 rent700, by(north, title(My title))
```

When using the `legend()` option combined with the `by()` option, we should place options that affect the position of the legend within the `by()` option. Consider this example:

```
. twoway scatter propval100 popden rent700,
         by(north, legend(pos(12))) legend(cols(1))
```

Here the `legend(pos(12))` option controls the position of the legend, placing it at the twelve o'clock position, so we place it within the `by()` option. On the other hand, the `legend(cols(1))` option does not affect the position of the legend, so we place it outside the `by()` option. For more details on this, see Options : By (section 8.8).

Altering the wrong axis When we use multiple x or y axes, it is easy to modify the wrong axis. Consider this example:

```
. twoway (scatter  propval100 ownhome)
         (scatter rent700 ownhome, yaxis(2) ytitle(Rents over 700))
```

We might think that the `ytitle()` option will change the title for the second y axis, but it will actually change the first axis. Because `ytitle()` is an option that concerns the overall graph, we should place it at the end of the graph command, as shown below.

```
. twoway (scatter propval100 ownhome)
         (scatter rent700 ownhome, yaxis(2)), ytitle(Rents over 700, axis(2))
```

We use the `axis(2)` option to indicate that `ytitle()` should be modified for the second y axis.

When all else fails I hope that by describing these errors you can avoid some common problems. Here are more ideas and resources to help you when you are struggling:

- Build graphs slowly. Rather than trying to make a final graph at once, try building the graph slowly by adding one option at a time. This is illustrated in Intro : Building graphs (section 1.6), where we took a complex graph and built it one piece at a time. Building slowly helps isolate problems for a particular option, which can then be investigated.
- When possible, model graphs from existing examples. This book strives to provide examples from which to model. For more online examples, see Introduction : Online supplements (section 1.1) for the companion web site for the book, which links to more examples.
- Reach out to fellow Stata users: colleagues, friends on Statalist (visit `https://www.stata.com/statalist/`), or Stata technical support (visit `https://www.stata.com/support/tech-support/`).

Subject index

A

`ac`, 11.1
`acprplot`, 11.1
added-variable plot, *see* avplot
`addplot()`, 11.3
  `msymbol()`, 11.3
adjacent lines, *see* alsize
`alsize`, 5.7
`alternate`, 5.4
alternate axes, *see* axes, alternate
angle, 10.1 , 10.1
  axis labels, *see* ylabel(); xlabel()
  label, 1.6 , 4.5 , 5.4
  marker labels, *see* mlabangle()
  marker symbol, 10.1 , 10.1 , 10.11 , 10.11
`angle0()`, 7.2
area graphs, *see* twoway area
  color, 2.5
  horizontal, 2.5
  setting the base, 2.5
  shading, 2.5
  sorting, 2.5
`ascategory`, 4.1 , 4.5 , 4.5 , 5.4 , 5.4 , 6.4
aspect ratio, 9.3
`aspectratio()`, 9.3
`asyvars`, 1.6 , 1.6 , 4.2 , 4.4 , 4.5 , 4.6 , 4.6 , 5.1 , 5.1 , 5.1 , 5.5 , 5.5 , 5.5 , 6.1 , 6.5
augmented component-plus-residual plot, *see* acprplot
`avplot`, 11.1 , 11.2 , 11.2
`aweight`, 2.1 , 8.1
axes
  alternate, 4.5 , 4.7 , 5.4 , 6.4 , 6.6 , 8.6
  bar graphs, 4.5 , 4.5 , 4.7 , 4.7

- base, 2.1
- box plots, 5.4 , 5.4 , 5.6 , 5.6
- categorical
  - bar graphs, 4.5 , 4.5
  - box plots, 5.4 , 5.4
  - dot plots, 6.4 , 6.4
  - titles, 1.6
- displaying for multiple graphs, 8.8
- dot plots, 6.4 , 6.4 , 6.6 , 6.6
- label gap, 4.5
- labels, *see* xlabel(); ylabel()
- lines, *see* xline(); yline()
- log scale, 8.6
- multiple, 2.10 , 2.11 , 2.11 , 3.2 , 3.2 , 3.2 , 8.4 , 8.7 , 8.7 , 8.10 , 8.10
- options, 8.4 , 8.4 , 8.7
- reverse scale, 8.6
- scale, *see* xscale(); yscale()
- scaling independently, 8.8
- selecting, 8.7 , 8.7
- size, 9.3 , 9.3
- suppressing, 4.7 , 5.6 , 8.6
- titles, *see* xtitle(); ytitle()

`axis()`, 3.2 , 8.4

B

`b1title()`, 1.6 , 1.6 , 4.5 , 5.4 , 8.8 , 9.1

`b2title()`, 4.5 , 5.4 , 9.1

`bar()`, 4.8
- `bcolor()`, 4.8
- `fcolor()`, 4.8 , 4.8
- `lcolor()`, 4.8 , 4.8
- `lwidth()`, 4.8 , 4.8

bar graphs, *see* twoway bar
- axes, *see* axes, bar graphs
- bar height, 4.6
- bar width, 2.6
- base, 2.6
- `by()`, 4.9 , 4.9

- categorical axes, *see* axes, categorical, bar graphs
- color, 4.8 , 4.8 , 4.8
- confidence intervals, 11.6
- descending, 4.4
- excluding missing bars, *see* nofill
- fill color, 2.6
- format, 1.6
- gaps, 4.3 , 4.3
- horizontal, *see* graph hbar
- labels, 1.6 , 4.6
- legend, 4.6 , 4.6
- line color, 2.6
- lines, 2.1
- look, 4.8 , 4.8
- ordering, 4.4 , 4.4
- overlaying, 4.3
- placing labels inside bars, 4.6 , 4.6
- reverse order, 4.4
- sorting, 4.4 , 4.4
- stacked, 4.1 , 4.2
- titles, 4.7
- vertical separators, 2.11 , 2.11
- y variables, 4.1 , 4.1

`barwidth()`, 2.6 , 2.7 , 2.8
`base()`, 2.1 , 2.5 , 2.6
`bcolor()`, 2.3 , 2.7
`bexpand()`, 8.11 , 8.11
`bin()`, 2.8
bins
- lower limit, 2.8
- number, 2.8

`blabel()`, 1.6 , 1.6 , 4.5 , 4.6 , 4.6
- `format()`, 1.6 , 1.6
- `gap()`, 4.6
- `position()`, 4.6 , 4.6

bold, 8.12 , 8.12
`box()`, 5.7 , 5.7 , 11.2

  - `bcolor()`, 5.7 , 5.7
  - `blcolor()`, 5.7
  - `blwidth()`, 5.7

box plots, *see* graph box
  - adjacent lines, *see* alsize
  - alphabetical order, 5.3
  - axes, *see* axes, box plots
  - `by()`, *see* by(), box plots
  - categorical axes, *see* axes, categorical, box plots
  - descending order, 5.3
  - excluding missing categories, *see* nofill
  - horizontal, *see* graph hbox
  - legend, 5.5 , 5.5
  - lines, 5.6
  - look, 5.7 , 5.7
  - median values, *see* medtype(); medmarker(); medline()
  - ordering, 5.3 , 5.3
  - `over()`, *see* over(), box plots
  - patterns, 5.6
  - sorting, 5.3
  - titles, 5.6
  - whiskers customized, *see* cwhiskers
  - *y* variables, 5.1 , 5.1

`boxgap()`, 5.2 , 5.7

bubble plots, 2.1 , 8.1 , 8.1

building a graph, 1.6 , 1.6

`by()`, 3.4 , 3.4 , 5.8 , 5.8 , 6.8 , 6.8 , 7.6 , 7.6 , 8.8 , 8.8 , 8.9 , 8.9
  - `alignment()`, 8.11
  - `b1title()`, 8.8
  - bar graphs, 4.9 , 4.9
  - `box`, 8.11 , 8.11
  - box plots, 5.8 , 5.8
  - `caption()`, 8.8
  - `colfirst`, 8.8
  - `cols()`, 4.9 , 5.8 , 6.8 , 8.8
  - combining options, 8.8 , 8.8
  - `compact`, 2.1 , 3.4 , 3.4 , 8.8

  - dot plots, 6.8 , 6.8
  - errors, 11.7
  - `height()`, 8.11
  - `holes()`, 8.8
  - `iscale()`, 8.8
  - `ixaxes`, 8.8 , 8.8
  - `ixtitle`, 8.8
  - `iyaxes`, 8.8
  - `iytitle`, 8.8
  - `justification()`, 8.11
  - `l1title()`, 8.8
  - `legend()`, 4.9 , 5.8 , 7.6 , 7.6 , 8.8 , 8.9 , 8.9
    - `at()`, 7.6 , 8.9
    - `position()`, 4.9 , 5.8 , 7.6 , 8.8
  - `missing`, 4.9 , 4.9 , 5.8 , 6.8
  - `noedgelabel`, 8.8
  - `note()`, 4.9 , 8.8
    - `suffix`, 8.8
  - pie charts, 7.6 , 7.6
  - `position()`, 8.11 , 8.11
  - `rescale`, 8.8
  - `ring()`, 8.11 , 8.11
  - `rows()`, 5.8 , 8.8
  - `scale()`, 3.4
  - scatterplot matrices, 3.4 , 3.4 , 8.8 , 8.8
  - `sts graph`, 11.2
  - `subtitle()`, 8.8
  - textboxes, 8.11 , 8.11
  - `title()`, 8.8 , 8.8 , 8.8
    - `position()`, 8.8
    - `ring()`, 8.8
  - `total`, 2.1 , 4.9 , 5.8 , 5.8 , 6.8 , 8.8
  - twoway, 2.1 , 2.1 , 2.10
  - `width()`, 8.11
  - `xrescale`, 8.8
  - `yrescale`, 8.8
- `bydimension()`, 11.3

  `label()`, 11.3
`byopts()`, 11.3
  `caption()`, 11.3
  `ixaxes`, 11.3
  `note()`, 11.3
  `subtitle()`, 11.3
  `title()`, 11.3

C
caps, *see* capsize()
`capsize()`, 5.7
`caption()`, 8.8 , 9.1 , 11.3
categorical axes, *see* axes, categorical
`ccolors()`, 2.9 , 2.9
`ccuts()`, 2.9
`ciopts()`, 11.3
  `lwidth()`, 11.3
  `msize()`, 11.3
`ciplot`, 2.3
clock position, 10.3 , 10.3
CMYK colors, 10.2.5
color
  area graphs, 2.5
  bar fill, 2.6
  bar graphs, 4.8 , 4.8
  bar lines, 2.1 , 2.6
  box plots, 5.7 , 5.7 , 5.7
  CMYK, 10.2.5
  confidence level, 2.3
  connecting lines, 2.4 , 2.7 , 8.3
  graph region, *see* graphregion()
  histogram bars, 2.8
  HSB, 10.2.5
  HSL, 10.2.5
  HSV, 10.2.5
  intensity, 4.8 , 5.7 , 10.2.2 , 10.2.2
  labels, 4.5 , 5.4

  - legend, 8.9 , 8.9
  - lines, 2.8 , 6.6 , 6.7
  - marker fill, 2.1 , 8.1
  - marker outline, 8.1
  - marker symbols, 2.1 , 2.1
  - markers, 2.4 , 2.7 , 6.7 , 6.7 , 6.7 , 8.1 , 8.1
  - median line, 5.7
  - named styles, 10.2.1 , 10.2.1
  - opacity, 10.2.3 , 10.2.3
  - overlap, 10.2.4 , 10.2.4
  - pie charts, 7.3 , 7.3
  - plot region, *see* plotregion()
  - RGB, 10.2.5
  - styles, 10.2 , 10.2.5
  - textbox, 8.11
- `cols()`, 11.4 , 11.4
- columns, 7.5
- combining graphs, 11.4 , 11.4
- commas with graph options, 11.7
- compass direction, 10.4 , 10.4
- component-plus-residual plot, *see* cprplot
- confidence interval
  - fit (regression predictions), 2.3 , 2.3
  - for means and percentiles of survival time, *see* stci
  - selecting display command, 2.3
  - setting level, 2.3
- confidence level
  - color, 2.3
  - pattern, 2.3
  - width, 2.3
- confidence regions
  - overlapping, 2.3 , 2.3
- `connect()`, 2.1 , 2.1 , 2.4 , 2.10 , 8.3 , 8.3 , 10.5 , 10.5
- connect lines width, *see* lwidth()
- connected plots, *see* twoway connected
- connecting
  - lines, *see* lines, connecting

- points, 2.1 , 2.1
- styles, 10.5 , 10.5

contour plots, 2.9 , 2.9
correlogram, *see* ac; pac
`cprplot`, 11.1
cross-correlogram, *see* xcorr
`crule()`, 2.9 , 2.9
`cumsp`, 11.1
cumulative spectral distribution graph, *see* cumsp
`cwhiskers`, 5.7

D

dates, 2.4 , 2.4
density, *see* kdensity; twoway kdensity
`descending`, 4.4 , 7.2
`diagonal`, 3.2
- `fcolor()`, 3.2

`discrete`, 2.8
displaying named graphs, 11.4
distribution graphs, 11.1
distribution plots, 2.8 , 2.8
dot plots, *see* graph dot; twoway dot
- alphabetical order, 6.3
- axes, *see* axes, dot plots
- `by()`, *see* by(), dots plots
- categorical axes, *see* axes, categorical, dot plots
- descending order, 6.3
- excluding missing categories, *see* nofill
- legend, 6.5 , 6.5
- look, 6.7 , 6.7
- ordering, 6.3
- `over()`, *see* over(), dot plots
- reverse order, 6.3
- sorting, 6.3

`dots()`, 6.7
- `mcolor()`, 6.7
- `msize()`, 6.7

- `msymbol()`, 6.7

dropped-line plots, *see* twoway dropline

E

`ecolor()`, 2.9 , 2.9

`exclude0`, 4.7 , 4.7 , 6.6

exploding pie slices, 7.3

exporting graphs, 11.5 , 11.5

F

`fcolor()`, 2.5 , 2.6 , 2.7 , 2.8

FEVD, *see* forecast-error variance decomposition

fits (regression predictions)
- fractional polynomial, *see* twoway fpfit; twoway fpfitci
- linear, *see* twoway lfit; twoway lfitci
- quadratic, *see* twoway qfit; twoway qfitci

fonts, 8.12 , 8.12

forecast-error variance decomposition, 11.1

forest plot, 11.1

formatting numbers
- bar labels, 1.6
- pie slices, 7.4 , 7.4

forty-five degree lines, 11.4

`fraction()`, 2.8

fractional polynomial fits, *see* twoway fpfit; twoway fpfitci

`frequency`, 2.8

function, line plot of, *see* twoway function

`fysize()`, 11.4

G

`gap()`, 2.8 , 2.8
- between boxes, 5.2
- between boxes and edge of plot, 5.2
- between columns, 8.9
- between labels and outside the graph, 4.5
- between labels and ticks, 5.4
- between lines, 6.2

  - between marker and label, 8.2
  - between rows, 8.9
  - box plots, 5.2 , 5.2
  - dot plots, 6.2
  - textboxes, 8.11
- `gladder`, 11.1
- `glcolor`, 8.5
- `graph bar`, 1.5 , 1.6 , 1.6 , 4 , 4.9
- `graph box`, 5 , 5.8
- `graph combine`, 11.4 , 11.4
- `graph display`, 1.6 , 11.4
- `graph dot`, 1.3 , 6 , 6.8
- `graph hbar`, 1.3 , 2.6 , 4 , 4.9
- `graph hbox`, 1.3 , 5 , 5.8
- `graph matrix`, 1.3 , 1.5 , 3 , 3.4
- `graph pie`, 1.3 , 7 , 7.6
- `graph save`, 11.4
- `graph use`, 11.4 , 11.4
- graphing a function, *see* twoway function
- `graphregion()`, 9.4 , 9.4
  - `color()`, 9.4
  - `fcolor()`, 9.4
  - `ifcolor()`, 9.4
  - `lcolor()`, 9.4
  - `lwidth()`, 9.4
- graphs, specialized, *see* specialized graphs
- Greek symbols, 8.12 , 8.12
- grids
  - displaying, 8.5 , 8.5
  - suppressing, 4.7 , 5.6 , 8.5
- groups, *see* by(); over()

H

- `half`, 3.3
- height
  - bar, 4.6
  - histogram bar, *see* histogram, bar height

- symbol, 8.9

hi-lo graphs, *see* range plots

histogram, *see* twoway histogram

- bar color, 2.8
- bar height, 2.8 , 2.8
- bar width, 2.8
- gap between bars, 2.8
- horizontal, 2.8
- opacity, 2.8 , 2.8
- overlapping, 2.8 , 2.8

`horizontal`, 2.1 , 2.5 , 2.6 , 2.7 , 2.8

HSB colors, 10.2.5

HSL colors, 10.2.5

HSV colors, 10.2.5

I

`if`, *see* samples, selecting

`imargin()`, 11.4

immediate graphs, *see* twoway scatteri

impulse–response function, 11.1

`in`, *see* samples, selecting

`intensity()`, 4.8 , 5.7

intensity, color, 10.2.2 , 10.2.2

IRF, *see* impulse–response function

`irf graph`, 11.1

IRT model, *see* item response theory model

`irtgraph`, 11.1

italics, 8.12 , 8.12

item information function, 11.1

item response theory model, 11.1

J

`jitter()`, 3.3

jittering, *see* scatterplot matrices, jittering

justification

- textboxes, 8.11 , 8.11
- titles, *see* title(), justification()

K

`kdensity`, 1.3 , 11.2

kernel density, *see* kdensity; twoway kdensity

  horizontal, 2.8

  lines, 2.8

  methods, 2.8

  opacity, 2.8 , 2.8

  overlapping, 2.8 , 2.8

`kernel(biweight)`, 2.8

L

`l1title()`, 4.5 , 5.4 , 6.4 , 8.8 , 9.1

`l2title()`, 4.5 , 5.4 , 6.4 , 9.1

`label()`, 8.9 , 11.2

labels, 1.5 , 1.5

  alternate, 4.5 , 5.4

  angles, *see* angle, label

  axes, *see* xlabel(); ylabel()

  bar graphs, 1.6 , 4.6

  changing, 4.5 , 4.5 , 4.6 , 5.4 , 5.5 , 6.4

  color, 4.5 , 5.4

  gap from axis, 4.5 , 5.4

  gap from outside edge of graph , 4.5

  gap from ticks, 5.4

  legend, *see* legend, labels

  marker symbols, 2.1 , 2.1

  markers, 2.10 , 3.1 , 8.2 , 8.2

  matrix, 3.2 , 3.2

  missing values, 4.5

  pie charts, 7.4 , 7.4 , 7.6

  placing inside bars, 4.6 , 4.6

  points, 8.10

  position, 4.6 , 4.6 , 7.4 , 8.10

  scale, 3.4

  size, 3.1 , 4.5 , 5.4 , 7.4

  suppressing, 3.4 , 4.5 , 4.6 , 5.5 , 5.5 , 6.5 , 8.5

  ticks, 5.4

  - time series, 2.4
  - titles, 2.10
- ladder of powers graphs, 11.1
- `lcolor()`, 2.1 , 2.3 , 2.4 , 2.5 , 2.6 , 2.7 , 2.8 , 8.3
- `legend()`, 1.5 , 1.5 , 1.6 , 1.6 , 2.1 , 2.1 , 2.1 , 2.10 , 2.11 , 2.11 , 2.11 , 4.6 , 4.6 , 4.6 , 4.9 , 5.5 , 5.5 , 5.8 , 6.5 , 6.5 , 7.4 , 7.5 , 7.5 , 7.6 , 8.8 , 8.9 , 8.9 , 11.2 , 11.3
  - bar graphs, 4.6 , 4.6
  - `bexpand`, 8.9
  - `bmargin()`, 8.9
  - `box`, 8.9
  - box plots, 5.5 , 5.5
  - `by()`, *see* by(), legend()
  - `colfirst`, 4.6 , 5.5 , 7.5 , 8.9 , 8.9
  - `colgap()`, 8.9
  - `color()`, 8.9
  - `cols()`, 1.5 , 2.1 , 2.10 , 4.6 , 4.6 , 4.9 , 5.5 , 5.5 , 5.8 , 6.5 , 7.5 , 7.6 , 8.9
  - columns, 4.6 , 4.6 , 4.9 , 5.5 , 5.5 , 7.5 , 8.9
  - dot plots, 6.5 , 6.5
  - `fcolor()`, 8.9
  - `holes()`, 7.5 , 8.9
  - key, 8.9
  - `label()`, 1.5 , 1.5 , 2.1 , 2.1 , 2.11 , 2.11 , 2.11 , 4.6 , 4.9 , 5.5 , 6.5 , 7.5 , 8.8 , 8.8 , 8.9 , 8.9
  - labels, 4.9 , 6.5 , 8.9 , 8.9
  - margins, 8.9
  - `note()`, 8.9
  - options, 8.9 , 8.9
  - `order()`, 2.1 , 2.1 , 7.5 , 8.9 , 8.9
  - overlaid graphs, 2.11 , 2.11 , 2.11
  - pie charts, 7.5 , 7.5 , 7.6
  - placing within plot regions, 4.6
  - `pos()`, 11.3
  - `position()`, 1.5 , 1.6 , 1.6 , 1.6 , 4.6 , 4.6 , 4.9 , 5.5 , 5.5 , 7.5 , 7.6 , 8.9 , 8.9
  - `region()`, 8.9
    - `fcolor()`, 8.9

        - `lcolor()`, 8.9
        - `lwidth()`, 8.9
        - `margin()`, 8.9
    - `ring()`, 1.6 , 1.6 , 4.6 , 4.6 , 8.9 , 11.3
    - `rowgap()`, 8.9
    - `rows()`, 1.6 , 1.6 , 1.6 , 4.6 , 5.5 , 6.5 , 7.5 , 8.9 , 8.9 , 8.9 , 11.3
    - `size()`, 8.9
    - `span`, 8.9
    - `stack`, 4.6 , 4.9 , 5.5 , 5.8 , 7.5 , 7.6 , 8.9
    - stacked, 4.6 , 5.5 , 8.9
    - `subtitle()`, 4.9 , 11.3
        - `bexpand`, 8.9
        - `box`, 8.9
    - suppressing, 4.6 , 7.4
    - `symxsize()`, 8.9
    - `symysize()`, 8.9
    - text, 8.9 , 8.9
    - `textfirst`, 4.6 , 8.9
    - `title()`, 6.5 , 7.5 , 7.5 , 8.9
        - `color()`, 8.9
        - `position()`, 7.5 , 7.5
        - `size()`, 8.9
    - titles, 8.9 , 8.9
    - twoway, 2.10
    - width, 8.9
- `level()`, 2.3
- `levels()`, 2.9
- leverage-versus-squared-residual plot, *see* lvr2plot
- life tables for survival data, *see* ltable
- `line()`, 7.3
    - `lcolor()`, 7.3
    - `lwidth()`, 7.3
- line plots, *see* twoway line
    - sorting, 2.4
- `line`, `twoway`, *see* twoway line
- linear fits, *see* twoway lfit; twoway lfitci
- linear regression diagnostics graphs, 11.1

`linegap()`, 6.2 , 6.2
`lines()`, 5.7 , 6.7
    adjacent, 5.7
    axes, *see* yline(); xline()
    box plots, 5.6
    color, 2.8 , 5.6 , 6.6 , 6.7
    connecting, 2.3 , 2.4 , 2.7 , 2.10 , 8.3 , 8.3
    fit, 2.3 , 2.3 , 2.3
    gap between, 6.2
    graph region, *see* graphregion()
    `lcolor()`, 5.7 , 6.7
    `lwidth()`, 5.7 , 6.7
    median, 5.7
    overlaying, 2.11
    patterns, 2.1 , 2.8 , 5.6 , 6.6 , 6.7 , 10.6 , 10.6
    plot region, *see* plotregion()
    styles, 2.11
    textbox outlines, 8.11 , 8.11
    whiskers, *see* cwhiskers
    width, 2.8 , 2.11 , 5.6 , 6.6
`linetype()`, 6.7 , 6.7
loading graphs, *see* graph use
local linear smooth plots, *see* twoway lowess
`lpattern()`, 2.3 , 2.4 , 2.7 , 2.8 , 2.11 , 8.3 , 10.6 , 10.6 , 11.2
`lroc`, 11.1
`lsens`, 11.1
`lstyle()`, 2.11
`ltable`, 11.1
`lvr2plot`, 11.1 , 11.2 , 11.2
`lwidth()`, 1.5 , 1.5 , 2.1 , 2.3 , 2.4 , 2.7 , 2.8 , 2.11 , 8.3 , 10.7 , 11.2

M
margins
    graph region, *see* graphregion()
    legend, 8.9
    plot region, *see* plotregion()
    plots, 11.3 , 11.3

  - styles, 10.8 , 10.8
  - textboxes, 8.11 , 8.11

- `marginsplot`, 11.3 , 11.3
  - opacity, 11.3 , 11.3
  - overlapping, 11.3 , 11.3
- `marker()`, 5.7 , 6.7 , 6.7
  - `mcolor()`, 6.7
  - `mfcolor()`, 6.7
  - `mlcolor()`, 6.7
  - `mlwidth()`, 6.7
  - `msize()`, 5.7 , 6.7 , 6.7
  - `msymbol()`, 5.7 , 6.7 , 6.7
- marker symbol angle, 10.1 , 10.1 , 10.11 , 10.11
- markers
  - box plots, 5.7
  - color, 2.4 , 2.7 , 3.1 , 3.1 , 6.7 , 6.7 , 6.7 , 8.1 , 8.1
  - displaying for data points, 2.4
  - fill color, 2.1 , 8.1
  - invisible, 8.1
  - label gap, 8.2
  - label size, 3.1
  - labels, 2.1 , 2.1 , 2.10 , 3.1 , 8.2 , 8.2
  - line width, 2.1
  - median line, 5.7
  - opacity, 2.1 , 2.1 , 2.11 , 2.11
  - options, 3.1 , 3.1 , 8.1 , 8.1 , 8.2 , 8.2
  - outline color, 8.1
  - outline width, 8.1
  - overlapping, 2.3 , 2.3
  - overlaying, 2.11
  - plus sign, 8.1
  - schemes, 8.1
  - size, 2.4 , 2.7 , 3.1 , 6.7 , 6.7 , 6.7 , 8.1 , 8.1 , 10.9 , 10.9
  - squares, 8.1 , 8.1
  - styles, 2.11 , 8.1 , 8.1 , 10.9 , 10.9 , 10.11 , 10.11
  - symbols, 1.5 , 1.5 , 2.1 , 2.1 , 2.10 , 2.11 , 3.1 , 3.1 , 10.11 , 10.11
  - width, 2.7

math symbols, 8.12 , 8.12
matrix
    axis labels, 3.2 , 3.2
    scatterplot, *see* scatterplot matrices
    titles, 3.2
`maxes()`, 3.2 , 3.4 , 3.4
    `xlabel()`, 3.2
    `xtick()`, 3.2
    `ylabel()`, 3.2 , 3.4 , 3.4
    `ytick()`, 3.2
`mcolor()`, 2.1 , 2.1 , 2.4 , 2.7 , 3.1 , 8.1
    opacity, 2.1 , 2.1 , 2.11 , 2.11 , 8.1
    overlapping, 2.3 , 2.3
`mean`, 4.1
`median`, 4.1 , 6.1
median band plots, *see* twoway mband
median line
    color, 5.7
    markers, 5.7
    width, 5.7
median points, *see* medtype(); medmarker(); medline()
median spline plots, *see* twoway mspline
`medline()`, 5.7
    `lcolor()`, 5.7
    `lwidth()`, 5.7
`medmarker()`, 5.7
    `msize()`, 5.7
    `msymbol()`, 5.7
`medtype()`, 5.7 , 5.7
`meta forestplot`, 11.1
meta-analysis, 11.1
`mfcolor()`, 2.1 , 2.1 , 3.1 , 8.1 , 8.1 , 8.1
`missing`, 4.2 , 4.5 , 5.1 , 6.1 , 7.1
`mlabangle()`, 8.2 , 10.1
`mlabcolor()`, 8.2
`mlabel()`, 2.1 , 2.1 , 2.10 , 3.1 , 8.1 , 8.2 , 8.2 , 8.10 , 10.1 , 10.12 , 11.2
`mlabgap()`, 8.2

`mlabposition()`, 2.1 , 2.1 , 8.1 , 8.2 , 10.3
`mlabsize()`, 2.1 , 3.1 , 8.2 , 8.2 , 10.12
`mlabvposition()`, 8.2 , 8.2
`mlcolor()`, 2.1 , 2.1 , 3.1 , 8.1 , 8.1
`mlwidth()`, 2.1 , 2.1 , 8.1 , 8.1
mountain plots, *see* twoway area
`msangle()`, 10.1 , 10.1 , 10.11 , 10.11
`msize()`, 2.1 , 2.1 , 2.4 , 2.7 , 3.1 , 8.1 , 8.1 , 8.1 , 10.9 , 10.9 , 11.2
`mstyle()`, 2.11 , 8.1 , 8.1
`msymbol()`, 1.5 , 1.5 , 1.5 , 2.1 , 2.1 , 2.1 , 2.4 , 2.7 , 2.10 , 2.11 , 3.1 , 3.1 , 8.1 , 8.1 , 8.1 , 10.3 , 10.11 , 10.11 , 11.2
multiple axes, *see* axes, multiple
multiple plots, *see* overlaying

N
`name()`, 11.4
named colors, 10.2.1 , 10.2.1
naming graphs, *see* name()
`ndots()`, 6.7
`noci`, 11.3
`noclockwise`, 7.2
`nofill`, 1.6 , 1.6 , 4.2 , 5.1 , 6.1
`nofit`, 2.3
`nogrid`, 11.2
`nolabel`, 4.5 , 4.6 , 5.5 , 6.5
`nooutsides`, 5 , 5.8
normal curve
    opacity, 2.8 , 2.8
    overlapping, 2.8 , 2.8
`note()`, 5.8 , 5.8 , 8.8 , 9.1 , 11.2 , 11.3

O
opacity
    color, 10.2.3 , 10.2.3
    confidence regions, 11.3 , 11.3
    histograms, 2.8 , 2.8
    kernel density, 2.8 , 2.8

  - `marginsplot`, 11.3 , 11.3
  - markers, 2.1 , 2.1 , 2.3 , 2.3 , 2.11 , 2.11
  - `mcolor()`, 2.1 , 2.1 , 2.3 , 2.3 , 2.11 , 2.11 , 8.1
  - normal curve, 2.8 , 2.8
  - scatterplots, 2.1 , 2.1 , 2.3 , 2.3 , 2.11 , 2.11
  - shading CIs, 2.3 , 2.3 , 2.11 , 2.11
  - `twoway function`, 2.8 , 2.8
  - `twoway histogram`, 2.8 , 2.8
  - `twoway kdensity`, 2.8 , 2.8
  - `twoway lfitci`, 2.3 , 2.3 , 2.11 , 2.11
  - `twoway qfitci`, 2.11 , 2.11
  - `twoway scatter`, 2.3 , 2.3
  - `twoway scatterplot`, 2.1 , 2.1 , 2.11 , 2.11
- options, 1.5 , 1.5
  - adding text, 8.10 , 8.10
  - axes, 8.4 , 8.4 , 8.7
  - labels, 1.5 , 1.5
  - legend, 1.5 , 1.5
  - marker symbols, 1.5 , 1.5
  - markers, 8.1 , 8.1 , 8.2 , 8.2
  - region, 9.4 , 9.4
  - scatterplot matrices, 3.3 , 3.3
  - specialized graphs, 11.2 , 11.2
  - standard, 9.4 , 9.4
  - textboxes, 8.11 , 8.11
  - titles, 1.5 , 1.5
  - using in the wrong context, 11.7
- ordering
  - bars, *see* bar graphs, ordering
  - boxes, *see* box plots, ordering
- orientation
  - textboxes, 8.11
  - titles, 10.10 , 10.10
- `outergap()`, 5.2
- outside values
  - color, 5.7
  - size, 5.7

  - suppressing, 5.6
- `over()`, 1.6 , 1.6 , 4.2 , 4.2 , 4.3 , 4.3 , 4.4 , 4.4 , 4.5 , 4.5 , 4.7 , 4.7 , 4.8 , 4.8 , 5 , 5.8 , 6 , 6.8 , 7 , 7.6
  - `asyvars`, 5.1 , 5.1
  - `axis()`, 4.5
    - `outergap()`, 4.5
  - bar graphs, 4.2 , 4.4 , 4.9 , 4.9
  - box plots, 5.1 , 5.1 , 6.1 , 6.3
  - `descending`, 4.4 , 5.3 , 5.3 , 6.3 , 6.3
  - display only existing variables, 4.2
  - dot plots, 6.1 , 6.3
  - `gap()`, 4.3 , 4.3 , 5.2 , 5.2 , 6.2
  - `label()`, 1.6 , 4.5 , 4.5 , 4.6 , 5.4 , 5.4
    - `angle()`, 1.6 , 4.5 , 5.4
    - `labcolor()`, 4.5 , 5.4
    - `labgap()`, 4.5 , 5.4 , 5.4
    - `labsize()`, 4.5 , 5.4
    - `ticks`, 4.5 , 5.4
    - `tlength()`, 4.5 , 5.4
    - `tlwidth()`, 4.5 , 5.4
    - `tposition()`, 4.5 , 5.4
  - missing, 4.2
  - pie charts, 7.1
  - `relabel()`, 4.5 , 4.5 , 5.4 , 5.4 , 6.4 , 6.4
  - `sort()`, 4.4 , 4.4 , 5.3 , 5.3 , 6.3 , 6.3
    - `sum()`, 4.4
  - y variables, 4.1 , 4.1
- overlapping
  - color, 10.2.4 , 10.2.4
  - confidence regions, 11.3 , 11.3
  - histograms, 2.8 , 2.8
  - `marginsplot`, 11.3 , 11.3
  - markers, 2.3 , 2.3
  - `mcolor()`, 2.3 , 2.3
  - normal curve, 2.8 , 2.8
  - reference line, 11.3
  - scatterplots, 2.3 , 2.3

  - shading CIs, 2.3 , 2.3
  - `twoway function`, 2.8 , 2.8
  - `twoway kdensity`, 2.8 , 2.8
  - `twoway lfitci`, 2.3 , 2.3
  - `twoway scatter`, 2.3 , 2.3
- overlaying, 2.2 , 2.2 , 2.11 , 2.11
  - bar graphs, *see* bar graphs, overlaying
  - connected marker plots, 2.4
  - fits, CIs, smooths, and scatters, 2.2 , 2.3 , 2.11 , 2.11
  - histograms, *see* histogram, overlaying
  - kernel density, *see* kernel density, overlaying
  - legends, *see* legend, overlaid graphs
  - lines, *see* lines, overlaying
  - markers, *see* markers, overlaying
  - mixed plot types, 1.3 , 1.3 , 1.5 , 1.5
  - scatterplots, *see* scatterplots, overlaying

P

- `pac`, 11.1
- patterns
  - axis lines, 2.1
  - box plots, 5.6
  - confidence level, 2.3
  - connecting lines, 2.3 , 2.7 , 8.3
  - lines, 2.8 , 5.6 , 6.6 , 6.7 , 10.6 , 10.6
- `percent`, 2.8 , 7.4
- `percentages`, 4.1 , 4.1 , 4.2
- `pergram`, 11.1
- periodogram, *see* pergram; wntestb
- `pie()`, 7.3 , 7.6
  - `color()`, 7.3
  - `explode`, 7.3 , 7.6
- pie charts, *see* graph pie
  - adding text, 7.4 , 7.4
  - angles, 7.2
  - `by()`, *see* by(), pie charts
  - color, 7.3 , 7.3

  counterclockwise, 7.2
  descending order, 7.2
  exploding slices, 7.3 , 7.3
  labels, 7.4 , 7.4 , 7.6
  legend, 7.5 , 7.5 , 7.6
  `over()`, *see* over(), pie charts
  slices, 7.3 , 7.3
  sorting, 7.2 , 7.2 , 7.6
  titles, 7.5
  types, 7.1 , 7.1
`plabel()`, 7.4 , 7.4 , 7.6
  `color()`, 7.4
  `format()`, 7.4 , 7.4
  `gap()`, 7.4 , 7.4
  `legend()`, 7.4
  `name`, 7.6
  `size()`, 7.4
`plotdimension()`, 11.3
  `allsimplelabels`, 11.3
  `elabels()`, 11.3
  `labels()`, 11.3
  `nolabels`, 11.3
  `nosimplelabels`, 11.3
`plotopts()`, 11.3
  `clwidth()`, 11.3
  `msize()`, 11.3
  `msymbol()`, 11.3
`plotregion()`, 9.4 , 9.4
  `color()`, 9.4
  `lwidth()`, 9.4
`pnorm`, 11.1
points, connecting, *see* connecting points
population pyramid, 11.6 , 11.6
position
  labels, 2.1 , 4.6 , 4.6 , 7.4 , 8.10
  legend, 1.6 , 4.6 , 4.6 , 4.9 , 5.5 , 5.5 , 7.5 , 7.6 , 8.9 , 8.9
  marker labels, 8.2 , 8.2

  - standard options, 10.3 , 10.4
  - ticks, 4.5 , 8.5
  - titles, *see* title(), position()

prefix, *see* titles, prefix

`ptext()`, 7.4 , 7.4
  - `orientation()`, 7.4
  - `placement()`, 7.4

Q

`qladder`, 11.1

`qnorm`, 11.1

quadratic fits, *see* twoway qfit; twoway qfitci

R

`r1title()`, 9.1

`r2title()`, 9.1

range plots
  - with area shading, *see* twoway rarea
  - with bars, *see* twoway rbar
  - with capped spikes, *see* twoway rcap
  - with capped spikes and marker symbols, *see* twoway rcapsym
  - with connected lines, *see* twoway rconnected
  - with lines, *see* twoway rline
  - with markers, *see* twoway rscatter
  - with spikes, *see* twoway rspike

`recast()`, 11.3

`recastci()`, 11.3

receiver operating characteristic analysis, 11.1

`rectangles()`, 6.7
  - `fcolor()`, 6.7
  - `lcolor()`, 6.7

reference line, overlapping, 11.3

reference lines, *see* lines, axes; yline()

region, 9.4 , 9.4

`replace`, 11.4

rescheming graphs, *see* scheme()

residual-versus-fit plot, *see* rvfplot

residual-versus-predictor plot, *see* rvpplot
restoring graphs, *see* graph use
reusing graphs, 11.4 , 11.4
reversing axes, *see* axes, reverse scale
RGB colors, 10.2.5
ROC analysis, *see* receiver operating characteristic analysis
`roccomp`, 11.1
`rocplot`, 11.1
`roctab`, 11.1
`rvfplot`, 11.1 , 11.2
`rvpplot`, 11.1
`rwidth()`, 6.7

S
samples, selecting, 2.4
`saving()`, 11.4
saving graphs, 11.4 , 11.4
`scale()`, 3.3 , 5.6 , 9.3 , 9.3 , 11.2
    adjusting, 9.3 , 9.3
    axes, 2.10 , 6.6 , 8.6 , 8.6
    labels, 3.4
    markers, 3.4
scatter with immediate arguments, *see* twoway scatteri
`scatter`, `twoway`, *see* twoway scatter
scatterplot matrices, 1.3
    `by()`, *see* by(), scatterplot matrices
    displaying lower half, 3.3
    jittering, 3.3
    options, 3.3 , 3.3
    scale, 3.3
scatterplots, *see* twoway scatter
    opacity, 2.1 , 2.1 , 2.11 , 2.11
    overlapping, 2.3 , 2.3
    overlaying, 2.1 , 2.1 , 2.10 , 2.10
`scheme()`, 11.3 , 11.4 , 11.4
schemes, 1.4 , 1.4.4 , 9.2 , 9.2.7
    `538`, 1.4.2 , 9.2.4 , 9.2.5

  - `538bw`, 9.2.4 , 9.2.5
  - `538w`, 1.4.2 , 9.2.4 , 9.2.5
  - customizing, 9.2.6 , 9.2.6
  - `economist`, 1.4.1 , 9.2.4 , 9.2.5 , 11.2 , 11.3
  - `lean1`, 1.4.2 , 9.2.4 , 9.2.5
  - `lean2`, 1.4.2 , 9.2.4 , 9.2.5
  - markers, 8.1
  - `plotplain`, 1.4.2 , 9.2.4 , 9.2.5
  - `plotplainblind`, 9.2.5
  - `plottig`, 1.4.2 , 9.2.4 , 9.2.5
  - `plottigblind`, 9.2.5
  - rescheming named graphs, *see* scheme()
  - `s1color`, 1.4.1 , 9.2.4 , 9.2.5
  - `s1mono`, 1.4.1 , 9.2.4 , 9.2.5 , 11.4
  - `s2color`, 1.4.1 , 9.2.4 , 9.2.5
  - `s2mono`, 1.4.1 , 9.2.4 , 9.2.5
  - `vg_hollowc`, 1.4.3 , 8.1
  - `vg_lgndc`, 1.4.3 , 4.2 , 4.6 , 5.5 , 5.6 , 7.2 , 7.5
  - `vg_palec`, 1.4.3 , 4.6 , 4.8
  - `vg_samec`, 1.4.3 , 2.1 , 8.1 , 11.6
- `scolor()`, 2.9 , 2.9
- separating graphs, 2.1 , 2.1
- shading area graphs, 2.5
- shading CIs
  - opacity, 2.11 , 2.11
  - overlapping, 2.3 , 2.3
- `showyvars`, 4.6 , 4.6 , 5.5 , 5.5
- size
  - adjacent line, 5.7
  - adjusting, 9.3 , 9.3
  - axes, 9.3 , 9.3
  - caps, 5.7
  - labels, 2.1 , 4.5 , 5.4 , 7.4
  - marker symbols, 2.1 , 2.1
  - markers, 2.4 , 2.7 , 3.1 , 3.1 , 6.7 , 6.7 , 6.7 , 8.1 , 8.1 , 10.9 , 10.9
  - text, 10.12 , 10.12
  - textbox, 4.7 , 8.11 , 8.11

- titles, 4.7 , 5.6

slices, 7.3 , 7.3

`sort`, 2.1 , 2.1 , 2.4 , 2.4 , 2.5 , 2.7 , 2.10 , 7.2 , 7.2 , 7.6 , 7.6 , 8.3 , 8.3 , 10.5 , 10.5

sorting
- area graphs, *see* area graphs, sorting
- box plots, *see* box plots, sorting
- line plots, *see* line plots, sorting
- pie charts, *see* pie charts, sorting

spacing, *see* gap

spaghetti plot, 11.6

specialized graphs, 11.1 , 11.1 , 11.2 , 11.2

spike plots, *see* twoway spike

splines, *see* twoway mspline

`stack`, 4.1 , 4.2 , 4.4 , 4.6

stacking bars, *see* stack

standard error of forecast, *see* stdf

standard options, 9.4 , 9.4

standardized normal probability graphs, 11.1

`start()`, 2.8

statistical function graphs, *see* twoway function

`stci`, 11.1

`stcoxkm`, 11.1

`stcurve`, 11.1

`stdf`, 2.3

`stepstair`, 8.3 , 10.5

storing graphs, *see* graph save

`stphplot`, 11.1

strip plots, 8.8 , 8.8 , 11.6

`sts graph`, 11.1 , 11.2

styles, 10 , 10.12
- angles, 10.1 , 10.1
- clock position, 10.3 , 10.3
- CMYK colors, 10.2.5
- color, 10.2 , 10.2.5
- color intensity, 10.2.2 , 10.2.2
- color opacity, 10.2.3 , 10.2.3

  - color overlap, 10.2.4 , 10.2.4
  - compass direction, 10.4 , 10.4
  - HSB colors, 10.2.5
  - HSL colors, 10.2.5
  - HSV colors, 10.2.5
  - lines, 2.11
  - margins, 10.8 , 10.8
  - marker symbols, 10.11 , 10.11
  - markers, 2.11 , 8.1 , 8.1 , 10.9 , 10.9 , 10.11 , 10.11
  - named colors, 10.2.1 , 10.2.1
  - orientation, 10.10 , 10.10
  - RGB colors, 10.2.5
  - text size, 10.12 , 10.12
- subscripts, 8.12
- `subtitle()`, 4.9 , 8.8 , 8.8 , 9.1 , 11.3
  - `nobexpand`, 4.9
  - `position()`, 4.9 , 8.8 , 8.8
  - `prefix`, 8.8 , 8.8
  - `ring()`, 4.9 , 8.8
  - `suffix`, 8.8 , 8.8
- suffix, *see* titles, suffix
- `sum`, 4.6
- superscripts, 8.12
- survival graphs, 11.1
- symbols
  - height, 8.9
  - margin, *see* msymbol()
  - width, 8.9
- symmetry plots, 11.1
- `symplot`, 11.1

T

- `t1title()`, 9.1
- `t2title()`, 9.1
- test information function, 11.1
- `text()`, 2.1 , 2.10 , 2.10 , 8.10 , 8.10 , 8.11 , 8.11 , 11.2
  - adding, 2.1 , 2.10 , 2.10 , 8.10 , 8.10

  - `box`, 2.1
  - `color()`, 8.11
  - legend, 8.9 , 8.9
  - `linegap()`, 8.11
  - `lwidth()`, 2.1
  - `margin()`, 2.1 , 8.11
  - `orientation()`, 3.2 , 8.11
  - pie charts, 7.4 , 7.4
  - `placement()`, 8.10 , 8.10 , 8.11 , 8.11
  - scale, 3.4
  - `size()`, 2.1 , 8.11 , 8.11 , 10.12 , 10.12
- textboxes, 8.11 , 8.11
  - `by()`, *see* by(), textboxes
  - color, 8.11
  - interline gaps, 8.11
  - justification, 8.11 , 8.11
  - margins, 8.11 , 8.11
  - orientation, 8.11
  - outline, 8.11 , 8.11
  - size, 4.7 , 8.11 , 8.11
- thickness, *see* width
- ticks
  - controlling, 8.5 , 8.5
  - labels, 5.4
  - length, 8.5
  - matrix, 3.2
  - position, 8.5
  - suppressing, 8.5
  - time series, 2.4
- time series
  - labels, 2.4
  - line plots, *see* twoway tsrline; twoway tsline
  - minor labels, 2.4
  - ticks, 2.4
  - titles, 2.4
- `tin()`, 2.4
- `title()`, 1.5 , 1.5 , 1.5 , 2.10 , 8.8 , 8.11 , 8.11 , 9.1 , 9.1 , 11.2 , 11.3

  - `bcolor()`, 8.11
  - `bexpand`, 9.1 , 9.1
  - `bmargin()`, 8.11
  - `box`, 1.5 , 1.5 , 2.10 , 8.11 , 8.11 , 9.1 , 9.1
  - `fcolor()`, 8.11
  - `justification()`, 8.11 , 8.11 , 9.1
  - `lcolor()`, 8.11
  - `lwidth()`, 8.11
  - `margin()`, 10.8 , 10.8
  - `nobox`, 1.5
  - `placement()`, 10.4
  - `position()`, 9.1
  - `ring()`, 9.1
  - `size()`, 1.5 , 1.5
  - `span`, 9.1
- titles, 1.5 , 1.5 , 4.5 , 5.4 , 9.1 , 9.1
  - axes, *see* xtitle(); ytitle()
  - bar graphs, *see* bar graphs, titles
  - box plots, *see* box plots, titles
  - categorical axes, *see* axes, categorical, titles
  - justification, *see* title(), justification()
  - legend, 8.9 , 8.9
  - matrix, 3.2
  - multiple lines, 9.1
  - orientation, 10.10 , 10.10
  - pie charts, 7.5
  - placing in a box, *see* title(), box
  - placing inside a plot region, *see* title(), placement()
  - position, *see* title(), position()
  - prefix, 8.4 , 8.8 , 8.8 , 11.2
  - size, *see* title(), size()
  - suffix, 8.4 , 8.8 , 8.8 , 11.2
  - time series, 2.4
  - width, 9.1 , 9.1
- `tlabel()`, 2.4
- `tline()`, 2.4
- `tmlabel()`, 2.4

`tmtick()`, 2.4
transparency; see opacity, 2.11
`ttext()`, 2.4
  `orientation()`, 2.4
`ttitle()`, 2.4
twoway
  adding text, 2.10 , 2.10
  `by()`, *see* by(), twoway
  graphs, 2 , 2.11
  legend, 2.10
  options, 2.10 , 2.10
  overlaying, 2.11 , 2.11
  titles, 1.5 , 2.10
`twoway area`, 1.3 , 2.5 , 2.5
`twoway bar`, 1.3 , 2.6 , 2.6
`twoway connected`, 1.3 , 2.4 , 2.4 , 2.11
`twoway contour`, 1.3 , 2.9 , 2.9
`twoway contourline`, 2.9 , 2.9
`twoway dot`, 1.3 , 2.1
`twoway dropline`, 1.3 , 2.1 , 2.1
`twoway fpfit`, 1.3 , 2.2
`twoway fpfitci`, 2.3 , 2.3
`twoway function`, 1.3
  opacity, 2.8 , 2.8
  overlapping, 2.8 , 2.8
`twoway histogram`, 1.3 , 2.8 , 2.8 , 11.1
  opacity, 2.8 , 2.8
  overlapping, 2.8 , 2.8
`twoway kdensity`, 1.3 , 2.8 , 2.8 , 11.1
  opacity, 2.8 , 2.8
  overlapping, 2.8 , 2.8
`twoway lfit`, 1.3 , 1.3 , 1.5 , 1.5 , 2.2 , 2.2 , 2.11 , 2.11 , 8.9 , 8.9
`twoway lfitci`, 1.3 , 2.3 , 2.3
  opacity, 2.11 , 2.11
  overlapping, 2.3 , 2.3
`twoway line`, 1.3 , 2.4 , 2.4 , 2.11 , 2.11 , 8.6 , 8.6
`twoway lowess`, 1.3 , 2.2

`twoway mband`, 1.3
`twoway mspline`, 1.3 , 2.2
`twoway qfit`, 1.3 , 2.2 , 2.11 , 2.11 , 8.9 , 8.9
`twoway qfitci`, 2.3 , 2.3 , 2.11 , 2.11
  opacity, 2.11 , 2.11
`twoway rarea`, 1.3 , 1.3 , 2.7 , 2.11 , 2.11
`twoway rbar`, 1.3 , 2.7 , 2.7 , 2.7
`twoway rcap`, 1.3 , 2.7 , 2.7 , 2.7
`twoway rcapsym`, 1.3 , 2.7
`twoway rconnected`, 1.3 , 2.7 , 2.7 , 2.7
`twoway rline`, 1.3 , 2.7
`twoway rscatter`, 1.3 , 2.7
`twoway rspike`, 1.3 , 2.7 , 8.6 , 8.6
`twoway scatter`, 1.3 , 1.3 , 1.5 , 1.5 , 2.1 , 2.1 , 2.2 , 2.2 , 2.3 , 2.3 , 2.3 , 2.10 , 2.10 , 2.11 , 2.11 , 3.4 , 8 , 8.12
  overlapping, 2.3 , 2.3
`twoway scatteri`, 2.1 , 2.1
`twoway scatterplot`
  opacity, 2.1 , 2.1 , 2.11 , 2.11
`twoway spike`, 1.3 , 2.1 , 2.1 , 2.11 , 2.11 , 11.1
`twoway tsline`, 1.3 , 2.4 , 2.4 , 2.11
`twoway tsrline`, 1.3 , 2.4
types of graphs, 1.3 , 1.3

U
using graphs, *see* graph use

V
VAR model, *see* vector autoregressive model
`varbasic`, 11.1
vector autoregressive model, 11.1

W
whiskers, *see* cwhiskers
`width()`, 2.8 , 2.8 , 2.8
  axes, 8.6
  bars, 2.6

  - box plot lines, 5.2 , 5.7
  - confidence level, 2.3
  - connecting lines, 2.3 , 2.4 , 2.7 , 8.3
  - histogram bars, 2.8
  - legend, 8.9
  - lines, 2.8 , 2.11 , 5.6 , 6.6
  - marker outline, 8.1
  - markers, 2.7
  - median line, 5.7
  - symbols, 8.9
  - ticks, 4.5
  - titles, 9.1 , 9.1
- `wntestb`, 11.1

X

- `xalternate`, 4.7 , 4.7 , 5.4 , 6.4
- `xaxis()`, 8.7 , 8.10
- `xcorr`, 11.1
- `xdimension()`, 11.3
- `xlabel()`, 1.5 , 1.5 , 2.1 , 2.1 , 3.2 , 3.2 , 8.5 , 8.5 , 8.6 , 10.1 , 11.3
  - `alternate`, 8.5
  - `angle()`, 8.5
  - `axis()`, 3.2 , 3.2
  - `grid`, 8.5
  - `labsize()`, 1.5 , 1.5
  - `nogrid`, 8.5
  - `valuelabels`, 8.5
- `xline()`, 2.1
- `xscale()`, 2.1 , 8.6 , 8.6 , 11.3 , 11.4
  - `lwidth()`, 8.6
  - `range()`, 11.3
- `xsize()`, 9.3 , 11.2 , 11.4
- `xtitle()`, 2.1 , 2.10 , 8.4 , 8.4 , 11.2 , 11.2 , 11.3
  - `box`, 8.4
  - `orientation()`, 10.10
  - `prefix`, 8.4
  - `size()`, 8.4 , 11.2

Y

y variables
  bar graphs, *see* bar graphs, *y* variables
  multiple, 4.1 , 4.1
`yalternate`, 4.5 , 4.7 , 5.6 , 6.6
`yaxis()`, 2.10 , 2.11 , 2.11 , 8.4 , 8.7 , 8.7
`ycommon`, 11.4
`ylabel()`, 1.6 , 1.6 , 2.1 , 2.1 , 2.10 , 2.10 , 3.2 , 3.2 , 4.7 , 4.7 , 5.6 , 5.6 , 6.6 , 8.5 , 8.5 , 8.6 , 8.7 , 8.7 , 11.2 , 11.3 , 11.4
  `angle()`, 1.6 , 1.6 , 11.3
  `axis()`, 3.2 , 3.2 , 8.7 , 8.7
  `glcolor`, 8.5
  `glpattern`, 8.5 , 8.5
  `glwidth`, 8.5
  `grid`, 2.1 , 8.5 , 8.5
  `labgap()`, 8.5
  `labsize()`, 8.5
  `nogrid`, 2.1 , 4.7 , 5.6 , 8.5
  `nolabel`, 8.5
  `noticks`, 8.5
  `tlength()`, 8.5
  `tlwidth()`, 8.5
  `tposition()`, 8.5
`yline()`, 2.1 , 4.7 , 4.7 , 5.6 , 6.6 , 11.2
  `lcolor()`, 4.7 , 4.7 , 5.6 , 6.6
  `lpattern()`, 4.7 , 4.7 , 5.6 , 6.6
  `lwidth()`, 4.7 , 4.7 , 5.6 , 6.6
`ymlabel()`, 8.5
  `glcolor()`, 8.5
  `glpattern()`, 8.5
  `grid`, 8.5
`ymtick()`, 8.5
  `tposition()`, 8.5
`yreverse`, 4.7 , 4.7 , 5.6 , 6.6
`yscale()`, 2.10 , 2.11 , 4.6 , 4.7 , 5.6 , 6.6 , 8.6 , 8.6 , 11.3
  `axis()`, 2.11 , 8.6
  `range()`, 2.11 , 4.6 , 8.6 , 8.6 , 11.3

`ysize()`, 9.3 , 11.2 , 11.4
`ytick()`, 8.5
  - `tposition()`, 8.5

`ytitle()`, 1.6 , 1.6 , 2.1 , 2.1 , 4.7 , 4.7 , 5.6 , 5.6 , 6.6 , 8.4 , 8.4 , 8.7 , 11.2 , 11.3
  - `axis()`, 8.7
  - `bexpand`, 4.7 , 5.6 , 6.6
  - `box`, 4.7 , 5.6 , 6.6
  - `fcolor()`, 6.6
  - `orientation()`, 10.10
  - `size()`, 2.1 , 4.7 , 5.6
  - `suffix`, 8.4

`yvaroptions()`, 4.5 , 4.5 , 4.5 , 5.2 , 5.3 , 5.4 , 5.4 , 6.4 , 6.4
  - `label()`, 4.5
  - `relabel()`, 4.5 , 4.5 , 5.4 , 5.4 , 6.4 , 6.4